

Лабораторна робота № 3

Робота з файловою системою за допомогою скриптів bash

План

1. Теоретичний матеріал
2. Завдання

1. Bash (Bourne-Again Shell) – це високорівневий інтерпретатор командної мови Unix-подібних операційних систем, що виконує команди, які вводяться або зі стандартного вхідного потоку (клавіатури), або зчитуються з файлу, який називається скриптом.

Файл скрипту – це звичайний текстовий файл, що містить послідовність команд bash, і для якого встановлені права на виконання поточним користувачем або його групою. Встановити права на виконання для конкретного файлу можна за допомогою команди **chmod**, наприклад, якщо файл скрипту має назву **script.sh**, то встановити право на виконання цього файлу для всіх користувачів можна командою: **chmod a+x script.sh**. Аналогічно для скасування такого права слід скористатися командою **chmod a-x script.sh**. Для запуску цього файлу на виконання слід скористатися наступною командою: **./script.sh**.

Приклад скрипту, що виводить вміст поточного каталогу на консоль (екран) (рис. 1):

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

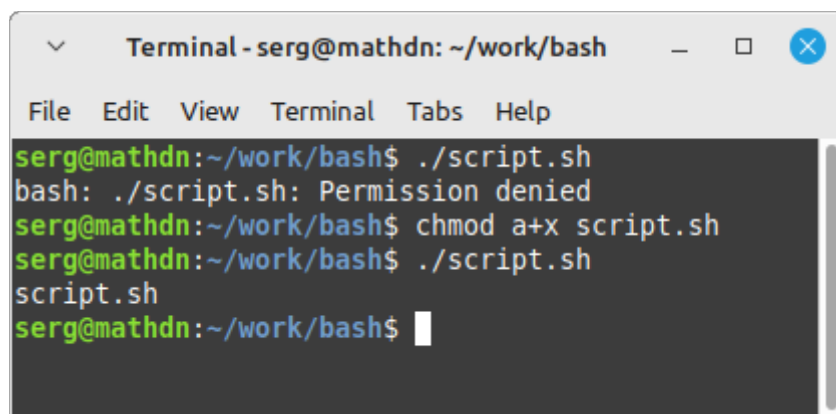


Рис. 1. Приклад запуску скрипту

Будь-який bash-скрипт повинен починатися з рядка, в якому вказується місце розташування командного інтерпретатора, наприклад, **#!/bin/bash**, де після символів **#!** безпосередньо вказується шлях до bash-інтерпретатора. Тому, якщо він встановлений в іншому каталозі, то цей шлях слід поміняти на актуальний в конкретній системі.

Коментарі у файлі скрипту починаються із символу **#** (окрім першого рядка, який, як вже зазначалося, є службовим).

У скрипті можуть використовуватися змінні. Ряд імен є зарезервованими:

– \$0, \$1, \$2, ... – значення аргументів командного рядка під час запуску скрипту, де \$0 – ім'я самого файлу скрипту, \$1 – його перший аргумент, \$2 – другий аргумент і т. д.;

– \$@ – всі аргументи командного рядка, кожен у лапках;

– \$# – кількість аргументів командного рядка;

– \$? – код повернення останньої команди.

Приклад простого скрипту, який виводить вміст каталогу, назва якого задана першим параметром скрипту, на консоль і у файл, ім'я якого задане другим параметром:

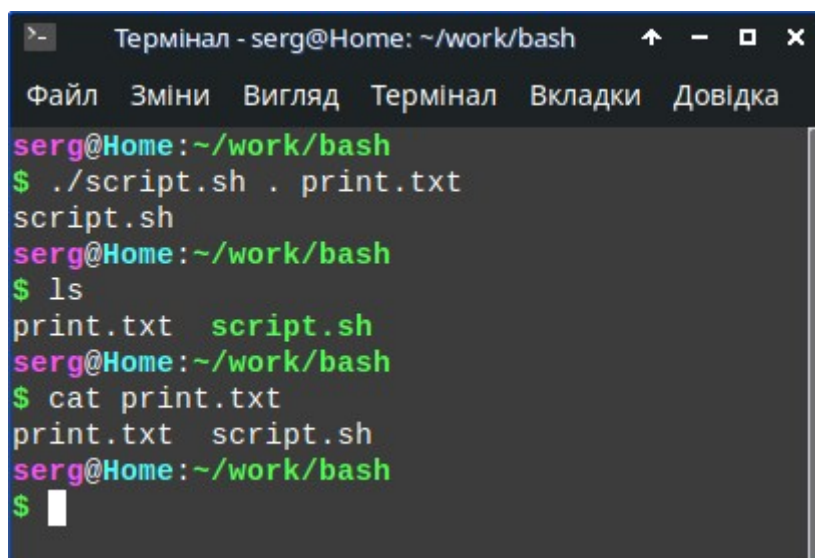
```
#!/bin/bash
```

```
dir $1
```

```
dir $1 > $2
```

```
.
```

Результат роботи цього скрипту наведено на рис. 2.

A screenshot of a terminal window titled "Термінал - serg@Home: ~/work/bash". The window shows the execution of a script named "script.sh" with two arguments: "." and "print.txt". The script's output is displayed in two parts. First, it lists the contents of the current directory, showing "print.txt" and "script.sh". Then, it cat's the contents of "print.txt", which also shows "print.txt" and "script.sh". The prompt "\$" is visible at the end of the last line.

```
>- Термінал - serg@Home: ~/work/bash
Файл  Зміни  Вигляд  Термінал  Вкладки  Довідка

serg@Home:~/work/bash
$ ./script.sh . print.txt
script.sh
serg@Home:~/work/bash
$ ls
print.txt  script.sh
serg@Home:~/work/bash
$ cat print.txt
print.txt  script.sh
serg@Home:~/work/bash
$
```

Рис. 2. Приклад виконання скрипту з параметрами

У скриптах можна створювати власні змінні, наприклад:

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

```
A=1
```

```
B="Hello world!"
```

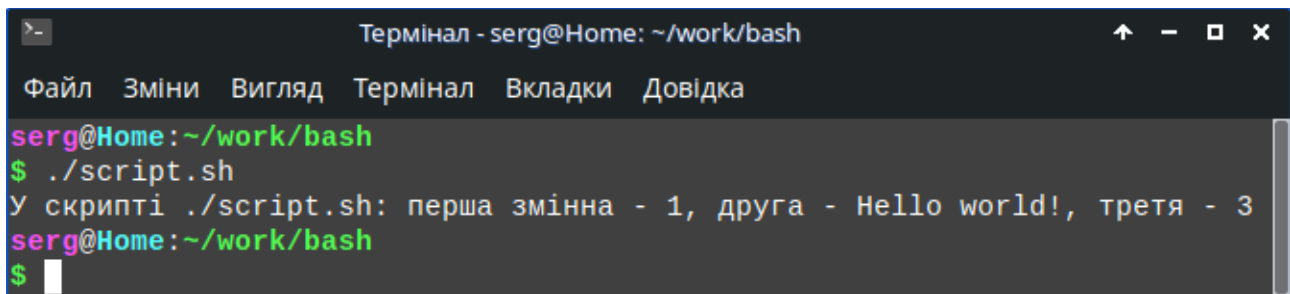
```
let C=3
```

```
echo "У скрипті $0: перша змінна - $A, друга - $B, третя - $C"
```

```
exit 0
```

Тут оператор **echo** використовується для виведення інформації на екран. Для отримання значення змінної використовується конструкція \$<ім'я_змінної>. У наведеному прикладі оператор echo виводить на екран рядок

з підстановкою значень змінних замість їхніх імен, оскільки рядок задається у подвійних лапках (рис. 3).



```
Термінал - serg@Home: ~/work/bash
Файл  Зміни  Вигляд  Термінал  Вкладки  Довідка
serg@Home:~/work/bash
$ ./script.sh
У скрипті ./script.sh: перша змінна - 1, друга - Hello world!, третя - 3
serg@Home:~/work/bash
$
```

Рис. 3. Виведення інформації з підстановкою значень змінних

Оператор **exit** використовується для повернення операційній системі результату роботи скрипту (0 – означає успішне завершення).

У скриптах можна використовувати умовні оператори. Найбільш поширеним типом останнього є структура **if-then-else**, яка має наступну форму:

if <умовний_оператор>

then

Оператори, що виконуються, якщо умовний оператор повертає 1

...

else

Оператори, що виконуються, якщо умовний оператор повертає 0

...

fi

У якості умовного оператора можуть виступати спеціальні структури **test**, **[[]]**, **[]**, **(())** або будь-яка інша **linux**-команда:

– **test** – використовується для логічного порівняння;

– **[]** – синонім команди **test**;

– **[[]]** – розширена версія **[]**, всередині якої можуть бути використані логічні зв'язки **||** (або), **&&** (та);

– **(())** – математичні порівняння.

Для побудови багаторівневих умов можна використовувати наступну форму умовного оператора: **if-then-elif-then-elif-....** Наприклад:

```
#!/bin/bash
echo -n "Enter your age: "
read age
if (( $age>=0 && $age<14 ))
then
    echo "You are a child"
elif (( $age>=14 && $age<18 ))
then
    echo "You are a teenager"
elif (( $age>=18 ))
then
```

```
        echo "You are a dinosaur)"
fi
```

Тут оператор **read** використовується для зчитування даних із вхідного потоку (клавіатури).

Якщо необхідно порівнювати одну змінну з великою кількістю можливих значень, доцільніше використовувати оператор **case**. Наприклад:

```
#!/bin/bash
echo -n "Enter file extension: "
read ext

echo -n "This is "
case "$ext" in
    sh) echo "a shell script";;
    c | cpp | hpp | h) echo "a C/C++ source file";;
    png) echo "PNG image file";;
    txt) echo "a text file";;
    zip | rar | tar | 7z) echo "an archive file";;
    conf) echo "a configuration file";;
    py) echo "a Python script";;
    *) echo "unknown file type";;
esac
```

В умовних операторах можна використовувати наступні умови порівняння :

1) файли:

- **-f** – файл існує (**!-f** – не існує);
- **-d** – каталог існує;
- **-s** – файл існує і не пустий;
- **-r** – файл існує і доступний для читання;
- **-w** – файл існує і доступний для запису;
- **-x** – файл існує і доступний для виконання;
- **-h** – символічне посилання;

2) рядки:

- **-z** – порожній рядок;
- **-n** – не порожній рядок;
- **==** – дорівнює (**!=** – не дорівнює);

3) числа:

- **-eq** – дорівнює;
- **-ne** – не дорівнює;
- **-lt** – менше;
- **-le** – менше чи дорівнює;
- **-gt** – більше;
- **-ge** – більше чи дорівнює.

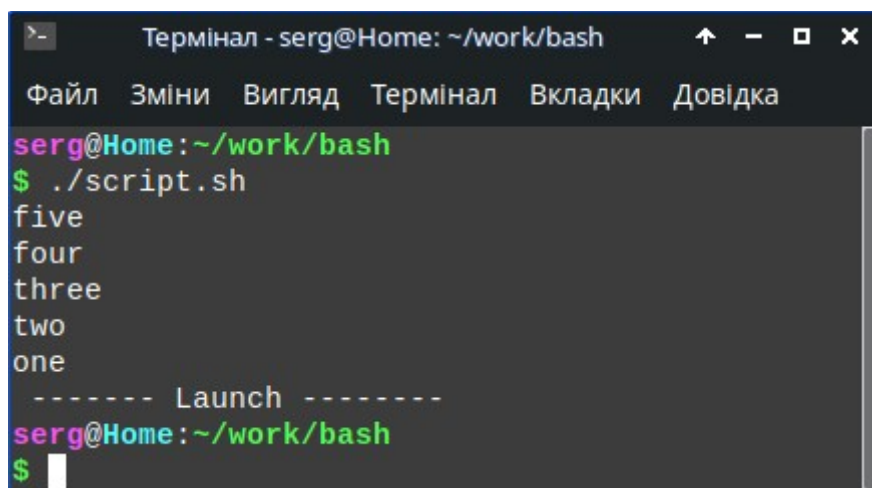
Наприклад:

```
#!/bin/bash
echo -n "Enter file name: "
read file_name

if [ -f $file_name ]
then
    echo "The file $file_name exists"
else
    echo "The file $file_name does not exist"
fi
```

У скриптах можуть використовуватися циклічні оператори. Найбільш простим серед них є **for-in**, в якому на кожній ітерації здійснюється звернення до чергового значення з наведеного списку, яке, в свою чергу, присвоюється цикловій змінній. Наприклад (рис. 4):

```
#!/bin/bash
for i in "five" "four" "three" "two" "one"
do
    echo $i
done
echo " ----- Launch -----"
exit 0
```



```
>- Термінал - serg@Home: ~/work/bash
Файл  Зміни  Вигляд  Термінал  Вкладки  Довідка
serg@Home:~/work/bash
$ ./script.sh
five
four
three
two
one
----- Launch -----
serg@Home:~/work/bash
$
```

Рис. 4. Приклад роботи циклу for-in

Обійти всі файли певного типу у заданому каталозі можна, наприклад, за допомогою такого скрипту:

```
#!/bin/bash
for f in $1/*; do
```

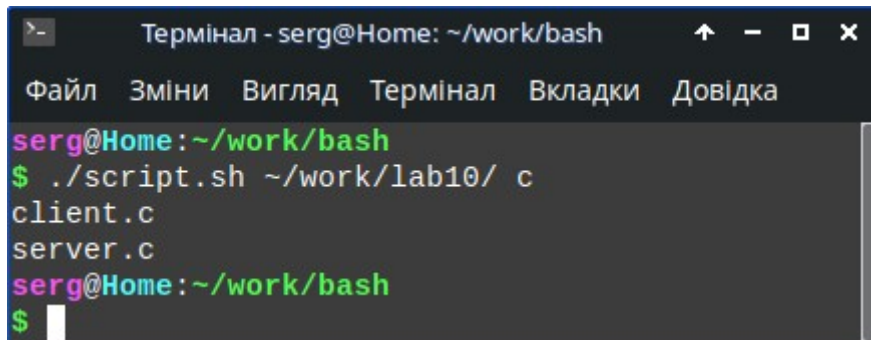
```

filename=$(basename "$f")
extension=${filename##*.}

if [ "$extension" == "$2" ]; then
    echo $filename
fi
done

```

Результат роботи цього скрипту наведено на рис. 5.



The screenshot shows a terminal window titled "Термінал - serg@Home: ~/work/bash". The window has a menu bar with options: "Файл", "Зміни", "Вигляд", "Термінал", "Вкладки", and "Довідка". The terminal content shows the user running a script: `serg@Home:~/work/bash`
`$./script.sh ~/work/lab10/ c`
`client.c`
`server.c`
`serg@Home:~/work/bash`
`$`

Рис. 5. Приклад роботи скрипту з обходу файлів у каталозі

Bash підтримує певний набір арифметичних операцій, які можна виконувати над числовими константами та змінними. Наприклад:

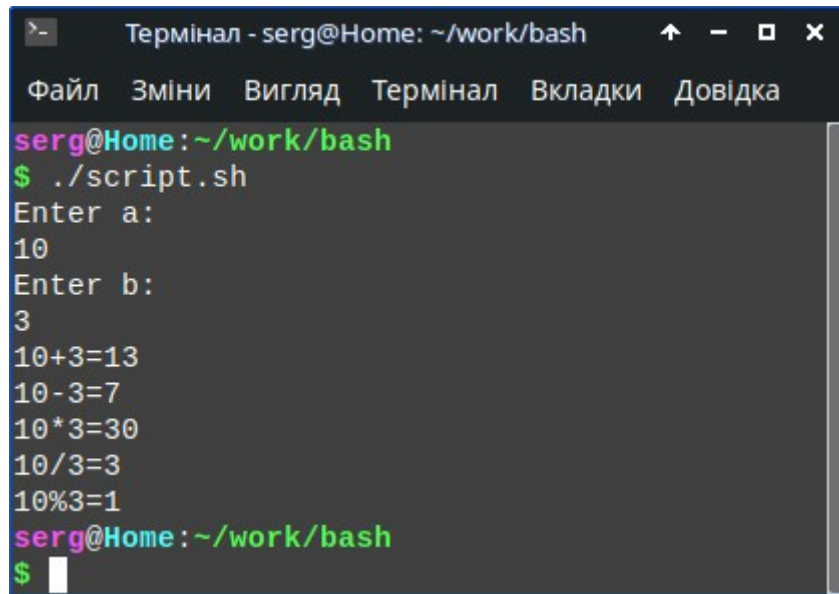
```

#!/bin/bash

echo "Enter a: "
read a
echo "Enter b: "
read b
let "c=a+b"
echo "$a+$b=$c"
let "c=a-b"
echo "$a-$b=$c"
let "c=a*b"
echo "$a*$b=$c"
let "c=a/b"
echo "$a/$b=$c"
let "c=a%b"
echo "$a%$b=$c"
exit 0

```

Після виконання цього скрипту буде отримано наступний результат (рис. 6).



```
>- Термінал - serg@Home: ~/work/bash
Файл  Зміни  Вигляд  Термінал  Вкладки  Довідка
serg@Home:~/work/bash
$ ./script.sh
Enter a:
10
Enter b:
3
10+3=13
10-3=7
10*3=30
10/3=3
10%3=1
serg@Home:~/work/bash
$
```

Рис. 6. Приклад виконання основних арифметичних операцій

У bash-скриптах можуть використовуватися функції, які можуть приймати аргументи і повертати числовий результат. Як і у мовах програмування, функції у скриптах дозволяють виключати дублювання коду та покращити його наочність.

Загальний синтаксис функції наступний:

```
function _name () {
    # commands
    ...
}
```

Розглянемо наступний приклад:

```
#!/bin/bash

search_file() {
    local dir_name="$1"
    local file_name="$2"

    for f in $dir_name/*
    do
        local f_name=$(basename "$f")

        if [ "$file_name"=="$f_name" ]
        then
            return 1
        fi
    done
}
```

```

        return 0
    }

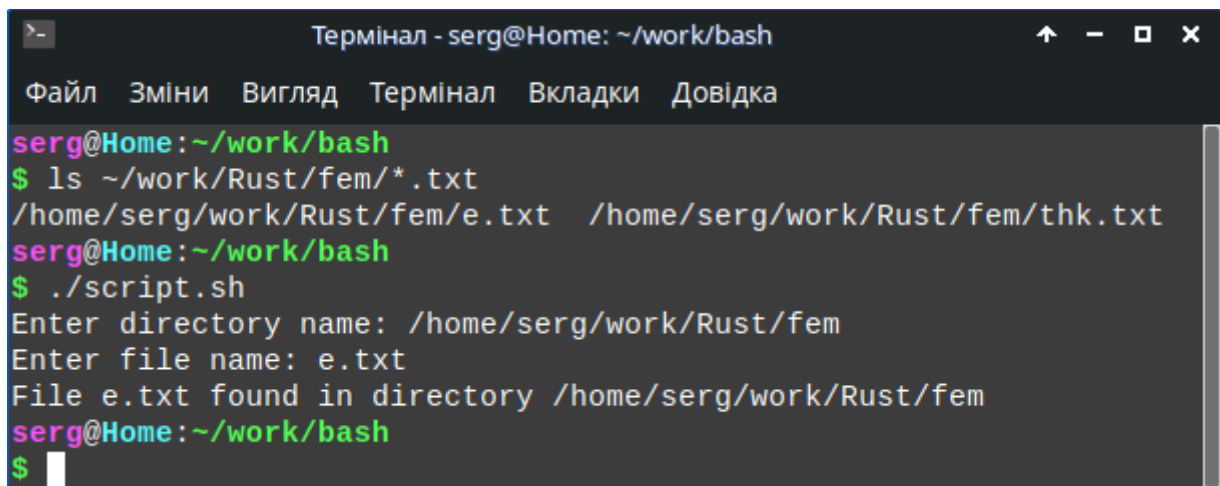
    echo -n "Enter directory name: "
    read dir_name
    echo -n "Enter file name: "
    read file_name

    search_file $dir_name $file_name # Виклик функції з параметрами
    func_result=$? # Отримання результату виконання функції
    if (( $func_result==0 ))
    then
        echo "File $file_name not found in directory $dir_name"
    else
        echo "File $file_name found in directory $dir_name"
    fi

```

В цьому скрипті реалізовано функцію `search_file()`, яка приймає два аргументи: `$1` – назва каталогу, де здійснюється пошук, і `$2` – назва шуканого файлу.

Для наочності за допомогою ключового слова **local** в тілі функції створюються дві локальні змінні, яким присвоюються значення параметрів. Якщо у вказаній теці знайдено шуканий файл, то за допомогою оператора **return** функція повертає значення 1, в іншому випадку – 0. Приклад виконання цього скрипту наведено на рис. 7.



```

Термінал - serg@Home: ~/work/bash
Файл  Зміни  Вигляд  Термінал  Вкладки  Довідка
serg@Home:~/work/bash
$ ls ~/work/Rust/fem/*.txt
/home/serg/work/Rust/fem/e.txt  /home/serg/work/Rust/fem/thk.txt
serg@Home:~/work/bash
$ ./script.sh
Enter directory name: /home/serg/work/Rust/fem
Enter file name: e.txt
File e.txt found in directory /home/serg/work/Rust/fem
serg@Home:~/work/bash
$

```

Рис. 7. Результат роботи скрипту з функцією

Файл, з якого здійснюється читання, називається **стандартним потоком введення**, а в який здійснюється запис – **стандартним потоком виведення**. Linux підтримує наступні стандарти потоки: 0 – `stdin` (стандартний потік вводу); 1 – `stdout` (стандартний потік виводу); 2 – `stderr` (стандартний потік помилок).

Для перенаправлення (переадресації) потоків використовуються спеціальні оператори `<`, `>`, `<<`, `>>` (табл. 1).

Таблиця 1. Найпоширеніші оператори переадресації

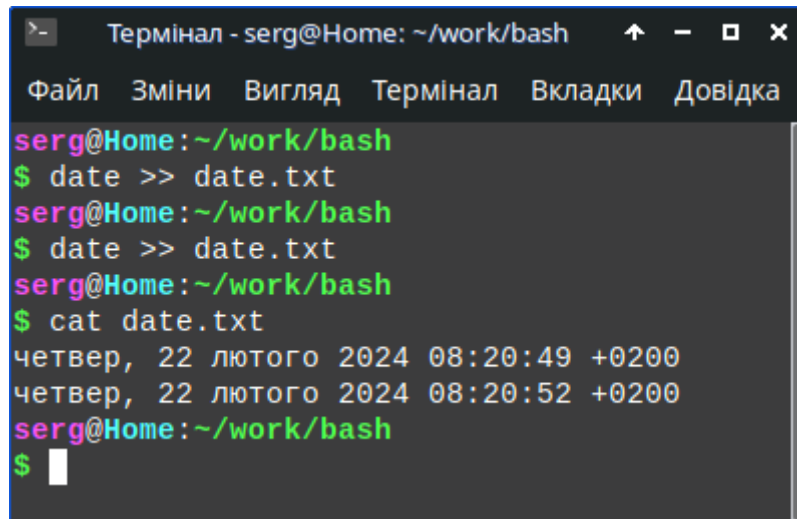
| № | Синтаксис | Опис |
|----------|---|--|
| 1 | команда <code>></code> файл | Направляє стандартний потік виводу команди у новий файл |
| 2 | команда <code>1></code> файл | Направляє стандартний потік виводу у вказаний файл |
| 3 | команда <code>>></code> файл | Направляє стандартний потік виводу у вказаний файл (режим приєднання) |
| 4 | команда <code>></code> файл <code>2>&1</code> | Направляє стандартні потоки виводу і помилок у вказаний файл |
| 5 | команда <code>2></code> файл | Направляє стандартний потік помилок у вказаний файл |
| 6 | команда <code>2>></code> файл | Направляє стандартний потік помилок у вказаний файл (режим приєднання) |
| 7 | команда <code>>></code> файл <code>2>&1</code> | Направляє стандартні потоки виводу і помилок у вказаний файл (режим приєднання) |
| 8 | команда <code><</code> файл1 <code>></code> файл2 | Отримує вхідні дані з файлу1 і направляє вихідні дані у файл2 |
| 9 | команда <code><</code> файл | В якості стандартного вхідного потоку команда отримує дані з вказаного файлу |
| 10 | команда <code><<</code> розділювач | Отримує дані зі стандартного потоку вводу до тих пір, поки не зустрінеться розділювач |
| 11 | команда <code><&m</code> | В якості стандартного вхідного потоку отримує дані з файлу з дескриптором <code>m</code> |
| 12 | команда <code>>&m</code> | Направляє стандартний потік виводу в файл з дескриптором <code>m</code> |

Слід зазначити, що оператор `n>&m` дозволяє перенаправляти файл з дескриптором `n` туди, куди спрямований файл з дескриптором `m`. Подібних операторів у командному рядку може бути декілька, в цьому випадку вони обчислюються зліва направо.

Крім того, у `bash`, починаючи з версії 3, можна використовувати оператор `<<<`, який зчитує дані з рядка замість файлу.

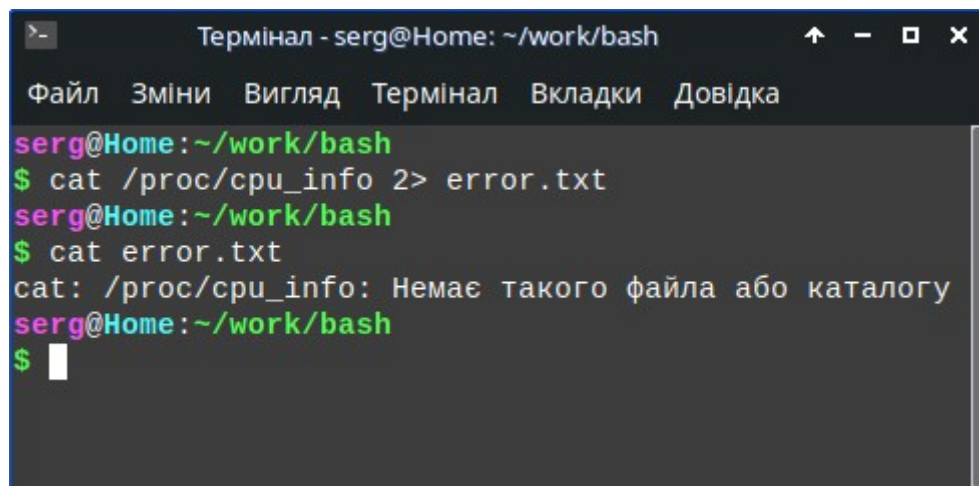
Наприклад, для виведення поточної дати та часу у файл в режимі приєднання (допису), можна виконати таку послідовність команд (рис. 8).

Щоб зафіксувати вміст `stderr`, слід використовувати перенаправлення `2>`. Більшість програм командного рядка виводять інформацію про помилки у стандартний канал помилок. Можна, наприклад, зафіксувати повідомлення про помилку, викликане спробою прочитання файлу, якого не існує (рис. 9).

A terminal window titled "Термінал - serg@Home: ~/work/bash" with a menu bar containing "Файл", "Зміни", "Вигляд", "Термінал", "Вкладки", and "Довідка". The terminal shows the following commands and output:

```
serg@Home:~/work/bash
$ date >> date.txt
serg@Home:~/work/bash
$ date >> date.txt
serg@Home:~/work/bash
$ cat date.txt
четвер, 22 лютого 2024 08:20:49 +0200
четвер, 22 лютого 2024 08:20:52 +0200
serg@Home:~/work/bash
$
```

Рис. 8. Перенаправлення виведення команди у файл в режимі приєднання

A terminal window titled "Термінал - serg@Home: ~/work/bash" with a menu bar containing "Файл", "Зміни", "Вигляд", "Термінал", "Вкладки", and "Довідка". The terminal shows the following commands and output:

```
serg@Home:~/work/bash
$ cat /proc/cpu_info 2> error.txt
serg@Home:~/work/bash
$ cat error.txt
cat: /proc/cpu_info: Немає такого файла або каталогу
serg@Home:~/work/bash
$
```

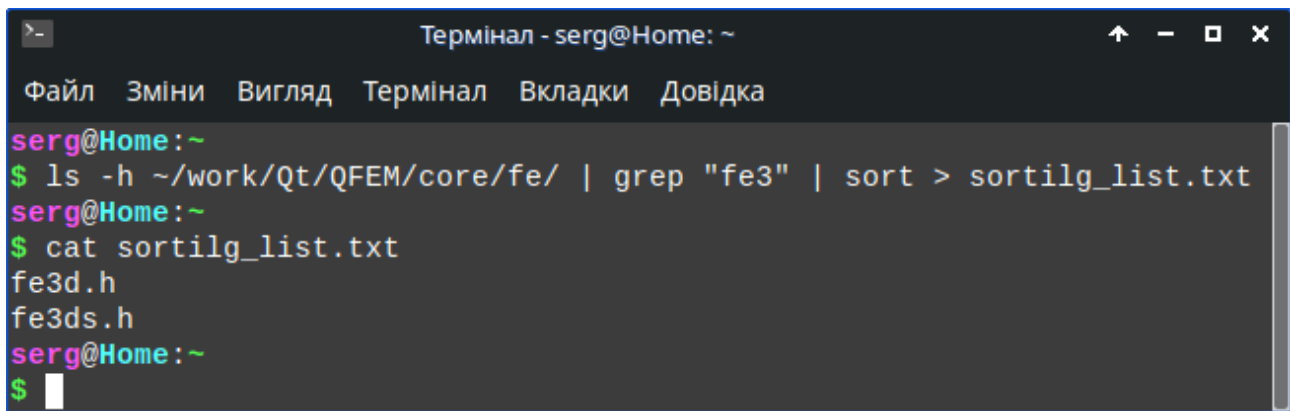
Рис. 9. Перенаправлення stderr

Дуже потужною можливістю bash є конвеєри, які дозволяють вивід однієї команди подавати на вхід у другу. Загальний синтаксис конвеєра наступний: команда1 | команда 2. Це значить, що вивід команди 1 передається на ввід команді 2.

Конвеєри можна групувати в ланцюжки і виводити за допомогою перенаправлення у файл, наприклад:

```
ls -h ~/work/Qt/QFEM/core/fe/ | grep "fe3" | sort > sortilg_list.txt
```

Тут команда ls виведе інформацію про вміст вказаної теки у зручній для користувача формі (ключ -h), після чого команда grep відбере в отриманому списку файли, назва яких містить підрядок "fe3", нарешті команда sort відсортує результат, після чого його буде перенаправлено у файл sortilg_list.txt (рис. 10).



```
Термінал - serg@Home: ~
Файл  Зміни  Вигляд  Термінал  Вкладки  Довідка
serg@Home:~
$ ls -lh ~/work/Qt/QFEM/core/fe/ | grep "fe3" | sort > sortilg_list.txt
serg@Home:~
$ cat sortilg_list.txt
fe3d.h
fe3ds.h
serg@Home:~
$
```

Рис. 10. Приклад використання конвеєра

2. Виконати завдання згідно з наведеними нижче варіантами.

Варіант 1. Написати скрипт для пошуку файлів заданого розміру у вказаному каталозі.

Варіант 2. Написати скрипт, який виводить на консоль розміри та права доступу для всіх файлів у заданому каталозі та всіх його підкаталогах.

Варіант 3. Написати скрипт пошуку заданого користувачем рядка у всіх файлах вказаного каталогу (з урахуванням підкаталогів).

Варіант 4. Написати скрипт, що обчислює сумарний розмір файлів у заданому каталозі та всіх його підкаталогах.

Варіант 5. Написати скрипт, що реалізує компіляцію та збирання всіх файлів початкового коду мовою C в заданому каталозі. Назву програми передавати у скрипт в якості параметра.