

Міністерство освіти і науки України
НТУУ «КПІ ім. Ігоря Сікорського»
Навчально-науковий інститут атомної та теплової енергетики
Кафедра цифрових технологій в енергетиці

Лабораторна робота №5
з дисципліни «Операційна система UNIX»
Тема «Створення сценаріїв в оболонці Bash»
Варіант №22

Студента 2-го курсу НН ІАТЕ гр. ТР-12

Ковальова Олександра

Перевірила: д.т.н., проф. Левченко Л. О.

Мета роботи. Набути навичок створювання bash-скриптів в ОС Linux. Освоїти базові конструкції мови.

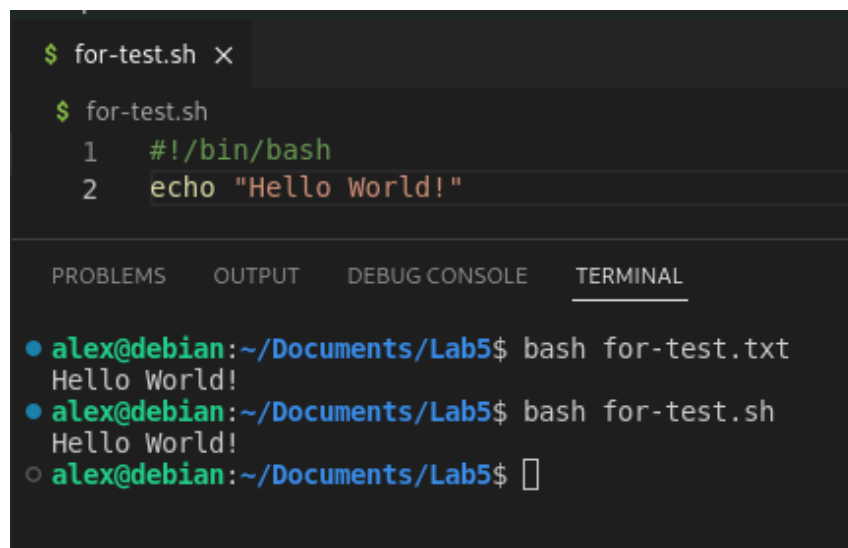
Теоретична частина. Bourne-again shell (GNU Bash) - це реалізація Unix shell, написана на С в 1987 році Брайаном Фоксом (Brian Fox) для GNU Project. Синтаксис мови Bash є надбудовою синтаксису мови Bourne shell. Переважна більшість скриптів для Bourne shell можуть бути виконані інтерпретатором Bash без змін, за винятком скриптів, які використовують спеціальні змінні або вбудовані команди Bourne shell. Також синтаксис мови Bash включає ідеї, запозичені з Korn shell (ksh) і C shell (csh).

Скрипт - це звичайний текстовий файл, що містить системні або вбудовані команди оболонки. Такий файл може бути запущений на виконання наступним чином:

```
bash script.sh
```

Оболонка послідовно інтерпретує і виконує команди, задані в сценарії. Ці ж команди можуть бути виконані простим послідовним викликом їх в командному рядку оболонки. Для файлів сценаріїв оболонка bash прийнято встановлювати розширення .sh.

Але – це не обов'язково. Linux в більшості не сприймає розширення файлів, все вирішують «біти доступу»[1]. Тому, мови які інтерпретуються, можна запускати використовуючи різні формати файлів:



```
$ for-test.sh ×  
$ for-test.sh  
1  #!/bin/bash  
2  echo "Hello World!"  
  
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  
● alex@debian:~/Documents/Lab5$ bash for-test.txt  
Hello World!  
● alex@debian:~/Documents/Lab5$ bash for-test.sh  
Hello World!  
○ alex@debian:~/Documents/Lab5$ □
```

Запускати скрипти можна безпосередньо вказуючи оболонку – цей варіант наведений вище. Але, також, можна зробити файл виконуваним за допомогою команди chmod:

```
chmod u+x file.sh
```

У цьому випадку, режим файлу змінений на виконуваний (x, executable) для власника файлу. Виконувані файли запускаються лише вказуючи поточну директорію:

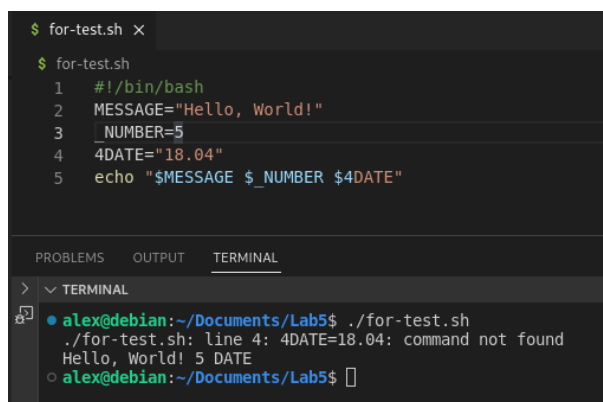
```
./file.sh
```

Це потрібно через проблеми з безпекою[2]. Всі вбудовані команди та утиліти знаходяться в певних директоріях, які, відповідно, знаходяться в змінній середовища \$PATH. Тобто, коли використовується якась утиліта, то оболонка перш за все шукає співпадіння там. Це зроблено для запобігання пасток. Наприклад, певний файл, який є шкідливим ПЗ, був перейменований в ls. Якщо викликати цю команду, то автоматично запустився б шкідливий файл. Але, на щастя, цю проблему вирішили ось таким способом.

За виконання програм, які можна запускати, відповідає загрузчик команд ядра. Загрузчик викликає системну функцію `exec()`, яка перевіряє перші 16 бітів файлу [3]. Якщо там є послідовність символів `#!` (She-bang, або шебанг)[2] – то програма читає рядок до кінця. Тому у багатьох виконуваних файлах ця послідовність знаходиться на першому рядку.

```
#!/bin/sh
#!/bin/bash
#!/usr/bin/perl
#!/usr/bin/tcl
#!/bin/sed -f
#!/usr/awk -f
```

Всі змінні у мові – текстові. Їх імена повинні починатися з літери і складатися з латинських букв, цифр і знаку підкреслення (`_`). Але – починати змінну можна лише з літери та знаку нижнього підчеркування[4]. Інші знаки, цифри, заброньовані ключові слова використовувати не можна. Щоб скористатися значенням змінної, треба перед нею поставити символ `$`. Використання значення змінної називається підстановкою.



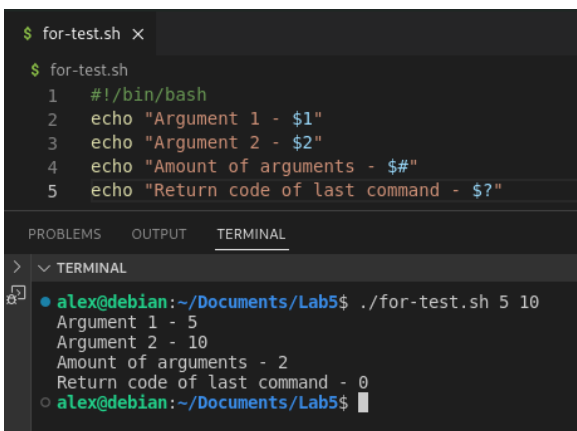
```
$ for-test.sh x
$ for-test.sh
1  #!/bin/bash
2  MESSAGE="Hello, World!"
3  _NUMBER=5
4  4DATE="18.04"
5  echo "$MESSAGE $_NUMBER $4DATE"

PROBLEMS  OUTPUT  TERMINAL
>  TERMINAL
● alex@debian:~/Documents/Lab5$ ./for-test.sh
./for-test.sh: line 4: 4DATE=18.04: command not found
Hello, World! 5 DATE
○ alex@debian:~/Documents/Lab5$
```

В мові програмування Bash існують позиційні змінні – для того, щоб можна було використовувати можливі аргументи передані в скрипт. Позначаються як знак долара та число (номер аргументу). Нумерація починається з 1. 0 аргумент – відносний шлях до скрипту. Приклад - `$0`, `$1`.

Спеціальна змінна `«$#»` повертає значення кількості аргументів. 0 аргумент не враховується.

Спеціальна змінна `«$?»` повертає код закінчення останньої операції (0 – успіх, все останнє – помилка).



```
$ for-test.sh x
$ for-test.sh
1  #!/bin/bash
2  echo "Argument 1 - $1"
3  echo "Argument 2 - $2"
4  echo "Amount of arguments - $#"
```

Для підстановки змінних використовується знак долару. Для підстановки команди може також використовуватись знак ``. Для арифметичних виразів потрібно використовувати подвійні дужки (не забуваючи про знак долару).

```
$ for-test.sh x
$ for-test.sh
1  #!/bin/bash
2  DATE=$(date)
3  PWD=$PWD
4  PWDCOMMAND=`pwd`
5  EXPR=$((5+5))
6  VAR=`abc`
7  echo "DATE: $DATE"
8  echo "PWD (using env variable): $PWD"
9  echo "PWD (using ``): $PWDCOMMAND"
10 echo "Arithmetic expression result: $EXPR"
11 echo "$VAR"
```

PROBLEMS OUTPUT TERMINAL

> TERMINAL

```
alex@debian:~/Documents/Lab5$ ./for-test.sh
./for-test.sh: line 6: abc: command not found
DATE: Wed 16 Nov 2022 11:40:04 PM EET
PWD (using env variable): /home/alex/Documents/Lab5
PWD (using ``): /home/alex/Documents/Lab5
Arithmetic expression result: 10
```

При перевірці певних даних, потрібно використовувати конструкцію if-then-else:

```
if [Умова 1]; then
    команда 1
    команда 2
elif [Умова 2]; then
    команда 3
    команда 4
else
    команда 5
fi
```

Квадратні дужки – це вбудована команда test (символічне посилання на неї). Тому, умова повинна бути відділена від дужок пробілами:

TEST(1)	User Commands	TEST(1)
NAME	test - check file types and compare values	
SYNOPSIS	<pre>test <u>EXPRESSION</u> test [<u>EXPRESSION</u>] [] [<u>OPTION</u></pre>	
DESCRIPTION	Exit with the status determined by EXPRESSION.	

Якщо потрібно використовувати логічні оператори такі як І, АБО, НЕ – то цей оператор не підходить. У такому випадку більш доцільно використовувати подвійні скобки [[]]. У цій інтерпретації команда test є зарезервованим ключовим словом.

Логічні оператори при використанні одинарних дужок можуть видавати помилки. З подвійними таких проблем немає. Але в цілому, якщо дані оператори не використовуються – то конструкція if-then-else-fi може працювати і без них.

```
$ for-test.sh X
$ for-test.sh
1  #!/bin/bash
2  NUMBER="5"
3  if [[ $NUMBER -eq 5 && $# -eq 0 ]]; then
4  |   echo "OK 1"
5  fi
6
7  if [ $NUMBER -eq 5 && $# -eq 0 ]; then
8  |   echo "OK 2"
9  fi
10
11 if cd /home/alex/ 2>/dev/null; then
12 |   echo "OK 3"
13 fi
```

PROBLEMS OUTPUT TERMINAL

> ▾ TERMINAL

```
● alex@debian:~/Documents/Lab5$ ./for-test.sh
OK 1
./for-test.sh: line 7: [: missing `]'
OK 3
○ alex@debian:~/Documents/Lab5$
```

Найпоширенішим циклом є конструкція while:

```
while <умова> do
    <оператори>
done
```

Приклад:

```
$ for-test.sh X
$ for-test.sh
1  #!/bin/bash
2  PASSWORD="pass"
3  while [ $PASSWORD != "passw0rd" ]; do
4  |   read -p "Enter password: " PASSWORD
5  done
6  echo "Success!"
```

PROBLEMS OUTPUT TERMINAL

> ▾ TERMINAL

```
● alex@debian:~/Documents/Lab5$ ./for-test.sh
Enter password: pass
Enter password: password
Enter password: passw0rd
Success!
○ alex@debian:~/Documents/Lab5$
```

Також, частовживаним є цикл for. Він має таку синтаксичну структуру:

```
for змінна in значення
do
    оператори
done
```

Приклад використання:

```
$ for-test.sh
1  #!/bin/bash
2  for FILE in *; do
3  |   echo $FILE
4  done
```

PROBLEMS OUTPUT TERMINAL

> ▼ TERMINAL

```
alex@debian:~/Documents/Lab5$ ./for-test.sh
for-test.sh
Kovalyov.txt
main.sh
v01.sh
v02.sh
v03.sh
v04.sh
v05.sh
v06.sh
```

Коли код повинен використовуватись багато разів, то доцільно записати його в функцію. Синтаксична конструкція функції:

```
function <ім'я> ()
{
    <СПИСОК>;
}
```

Приклад використання:

```
$ for-test.sh X
$ for-test.sh
1  #!/bin/bash
2  function directoryList() {
3  |   for FILE in *; do
4  |   |   echo $FILE
5  |   done
6  |   }
7
8  directoryList
```

PROBLEMS OUTPUT TERMINAL

> ▼ TERMINAL

```
v02.sh
v03.sh
v04.sh
v05.sh
v06.sh
v07.sh
v08.sh
```

Функції обов'язково оголошувати перед викликом. Причина очевидна: інтерпретатор виконує код рядково.

При виявленні помилки при виконанні сценарію командна оболонка виводить на екран номер рядка, що містить помилку. Якщо помилку відразу не видно, потрібно додати кілька додаткових команд echo для виведення значень змінних, протестувати фрагменти програмного коду, вводючи їх в командній оболонці в інтерактивному режимі. Основний спосіб відстеження помилок, які найбільш складно виявляються - використання опцій відладки командної оболонки.

Опції відладки командного рядка:

Опція	Призначення
sh -n <сценарій>	Тільки перевіряє синтаксичні помилки
sh -v <сценарій>	Виводить на екран команди перед їх виконанням
sh -x <сценарій>	Виводить на екран команди після обробки командного рядка
sh -u <сценарій>	Видає повідомлення про помилку при використанні невизначеної змінної

Приклад:

```
$ for-test.sh ×
$ for-test.sh
1  #!/bin/bash
2  if $# -eq 5; then
3      echo "OK"
4  fi

PROBLEMS  OUTPUT  TERMINAL
>  ▾ TERMINAL
● alex@debian:~/Documents/Lab5$ ./for-test.sh
./for-test.sh: line 2: 0: command not found
● alex@debian:~/Documents/Lab5$ bash -n for-test.sh
● alex@debian:~/Documents/Lab5$ bash -v for-test.sh
#!/bin/bash
if $# -eq 5; then
    echo "OK"
fi
for-test.sh: line 2: 0: command not found
● alex@debian:~/Documents/Lab5$ bash -u for-test.sh
for-test.sh: line 2: 0: command not found
○ alex@debian:~/Documents/Lab5$
```

Хід роботи

При виконанні роботи використовувався файловий менеджер mc:

```
Left  File  Command  Options  Right
<-  ~/Documents/Lab5  .[^>
.n  Name  Size  Modify time  .n  Name  Size  Modify time  .[^>
/..  UP--DIR  Nov 16 00:50  /..  UP--DIR  Oct 30 19:50
*main.sh  386  Nov 15 22:32  /.cache  4096  Nov 16 22:50
v01.sh  300  Nov 15 22:34  /.config  4096  Nov 14 22:46
v02.sh  104  Nov 15 22:35  /.gconf  4096  Oct 30 15:35
v03.sh  103  Nov 15 22:37  /.gnupg  4096  Nov 16 23:03
v04.sh  164  Nov 15 22:38  /.local  4096  Sep 14 15:17
v05.sh  553  Nov 15 22:41  /.mozilla  4096  Sep 23 18:53
v06.sh  100  Nov 15 22:42  /.pki  4096  Nov 14 22:46
v07.sh  133  Nov 15 22:42  /.ssh  4096  Sep 14 15:21
v08.sh  258  Nov 15 22:43  /.vscode  4096  Nov 14 22:46
v09.sh  396  Nov 15 22:54  /Desktop  4096  Nov 14 17:53
v10.sh  265  Nov 15 23:37  /Documents  4096  Nov 16 00:50
v11.sh  592  Nov 15 22:57  /Downloads  4096  Nov 14 22:28
v12.sh  453  Nov 15 23:10  /Music  4096  Sep 14 15:17
v13.sh  341  Nov 15 23:28  /Pictures  4096  Nov 15 02:14

v01.sh  10G/19G (55%)  UP--DIR  10G/19G (55%)
Hint: The homepage of GNU Midnight Commander: http://www.midnight-commander.org/
alex@debian:~/Documents/Lab5$
1Help 2Menu 3View 4Edit 5Copy 6RenMov 7Mkdir 8Delete 9PullDn10Quit
```

При відкритті файлів потрібно обрати текстовий редактор. В даній роботі використовувався vim:

```
#!/bin/bash

# Creating variable with name of the file
FILE="Kovalyov.txt"
{
    echo "Kovalyov Oleksandr Oleksiyovuch"
    echo "Academic group = TP-12"
    echo "Hobbies: Programming, learning Linux"
} | tee $FILE
# tee is a command in CLI using standard streams
# which reads standard input and writes it to both
# standard output and one or more files,
# effectively duplicating its input.
```

Але, для зручності, код, термінал та результати виконання будуть продемонстровані у редакторі коду Visual Studio Code.

Головним завданням було написати скрипт, який створює файл Kovalyov.txt. В цьому файлі повинна бути вказана така інформація як ПІБ, група, хобі.

```
$ main.sh x
$ main.sh
1  #!/bin/bash
2
3  # Creating variable with name of the file
4  FILE="Kovalyov.txt"
5  {
6      echo "Kovalyov Oleksandr Oleksiyovuch"
7      echo "Academic group = TP-12"
8      echo "Hobbies: Programming, learning Linux"
9  } | tee $FILE
10 # tee is a command in CLI using standard streams
11 # which reads standard input and writes it to both
12 # standard output and one or more files,
13 # effectively duplicating its input.
```

Для початку, оголошуємо змінну FILE, в якій вказана назва нового файлу. Потім створюємо так звану «анонімну функцію» – це потрібно для того, щоб, по-перше, не оголошувати зайвий раз функцію та не викликати її, а по-друге, щоб результат можна було передати в конвейер. Перенаправляємо результат трьох команд в утиліту tee, яка перенаправляє результат як на екран, так і у вказаний файл. Автоматично створюється файл Kovalyov.txt:

name	Size	Modify time
../	UP--DIR	Nov 16 00:50
Kovalyov.txt	92	Nov 17 00:55
*main.sh	386	Nov 15 22:32
v01.sh	300	Nov 15 22:34
v02.sh	104	Nov 15 22:35
v03.sh	103	Nov 15 22:37

Та виводиться інформація на екран:

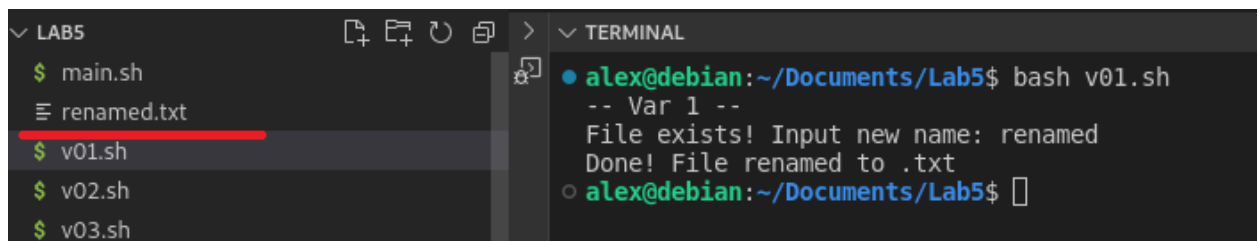
```
● alex@debian:~/Documents/Lab5$ ./main.sh
Kovalyov Oleksandr Oleksiyovuch
Academic group = TP-12
Hobbies: Programming, learning Linux
○ alex@debian:~/Documents/Lab5$
```

Окрім цього, потрібно було виконати індивідуальне завдання. Загалом було виконано всі 30 індивідуальних завдань.

Варіант 1: Сценарій перейменування власного файлу.

```
$ v01.sh
1  #!/bin/bash
2
3  echo "-- Var 1 --"
4
5  FILENAME="Kovalyov"
6
7  if [ -f $FILENAME.txt ]; then                # if file exists
8      read -p "File exists! Input new name: " NEWFILENAME
9      mv $FILENAME.txt $NEWFILENAME.txt        #renaming
10     echo "Done! File renamed to $NEWFILE.txt"
11 else
12     echo "File $FILENAME.txt doesn't exist"
13 fi
```

Алгоритм простий: якщо файл існує «ключ -f» то зчитуємо в змінну нову назву файлу (використовуємо команду read з ключем -p – тобто prompt з англ. – щоб користувач побачив запрошення до введення). Перейменовуємо файл за допомогою команди mv. Виводимо нову назву на екран. Якщо файл не існує, виводимо помилку.

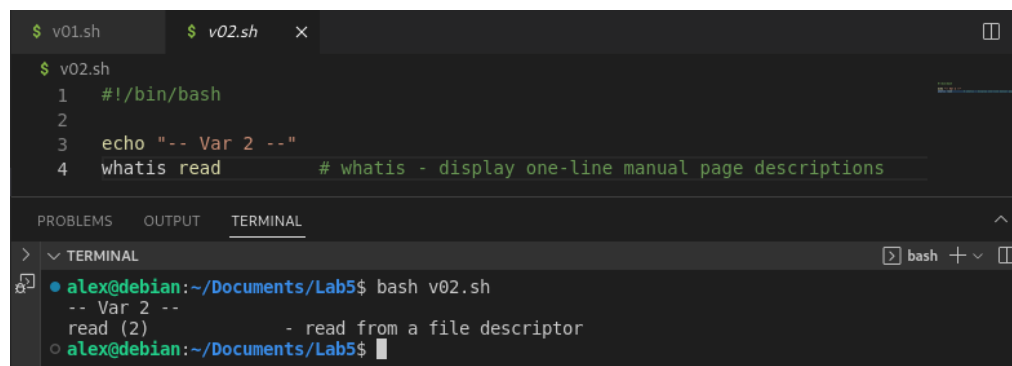


```
LAB5
$ main.sh
$ renamed.txt
$ v01.sh
$ v02.sh
$ v03.sh
```

```
● alex@debian:~/Documents/Lab5$ bash v01.sh
-- Var 1 --
File exists! Input new name: renamed
Done! File renamed to .txt
○ alex@debian:~/Documents/Lab5$
```

Варіант 2: Вивести коротку довідку щодо команди read.

Використовуємо команду whatis. За допомогою неї можна вивести опис команди в один рядок.



```
$ v01.sh
$ v02.sh
```

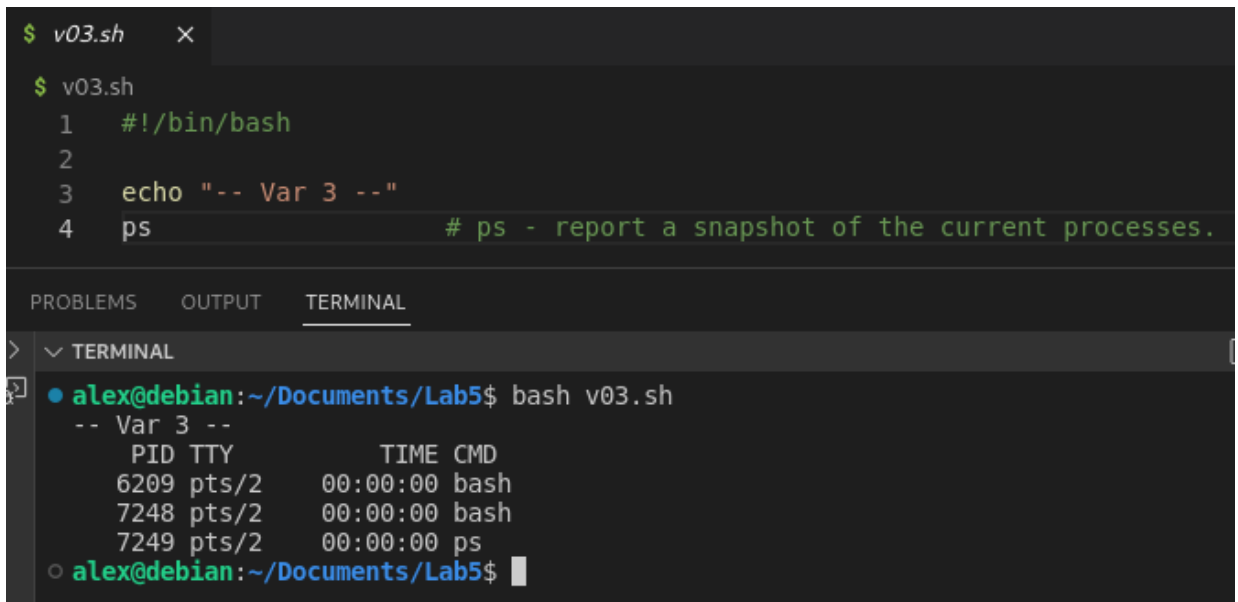
```
$ v02.sh
1  #!/bin/bash
2
3  echo "-- Var 2 --"
4  whatis read                                # whatis - display one-line manual page descriptions
```

```
PROBLEMS  OUTPUT  TERMINAL
```

```
> TERMINAL
● alex@debian:~/Documents/Lab5$ bash v02.sh
-- Var 2 --
read (2) - read from a file descriptor
○ alex@debian:~/Documents/Lab5$
```

Варіант 3: Відобразити список процесів.

Використовуємо команду ps.



```
$ v03.sh
$ v03.sh
1  #!/bin/bash
2
3  echo "-- Var 3 --"
4  ps                                # ps - report a snapshot of the current processes.
```

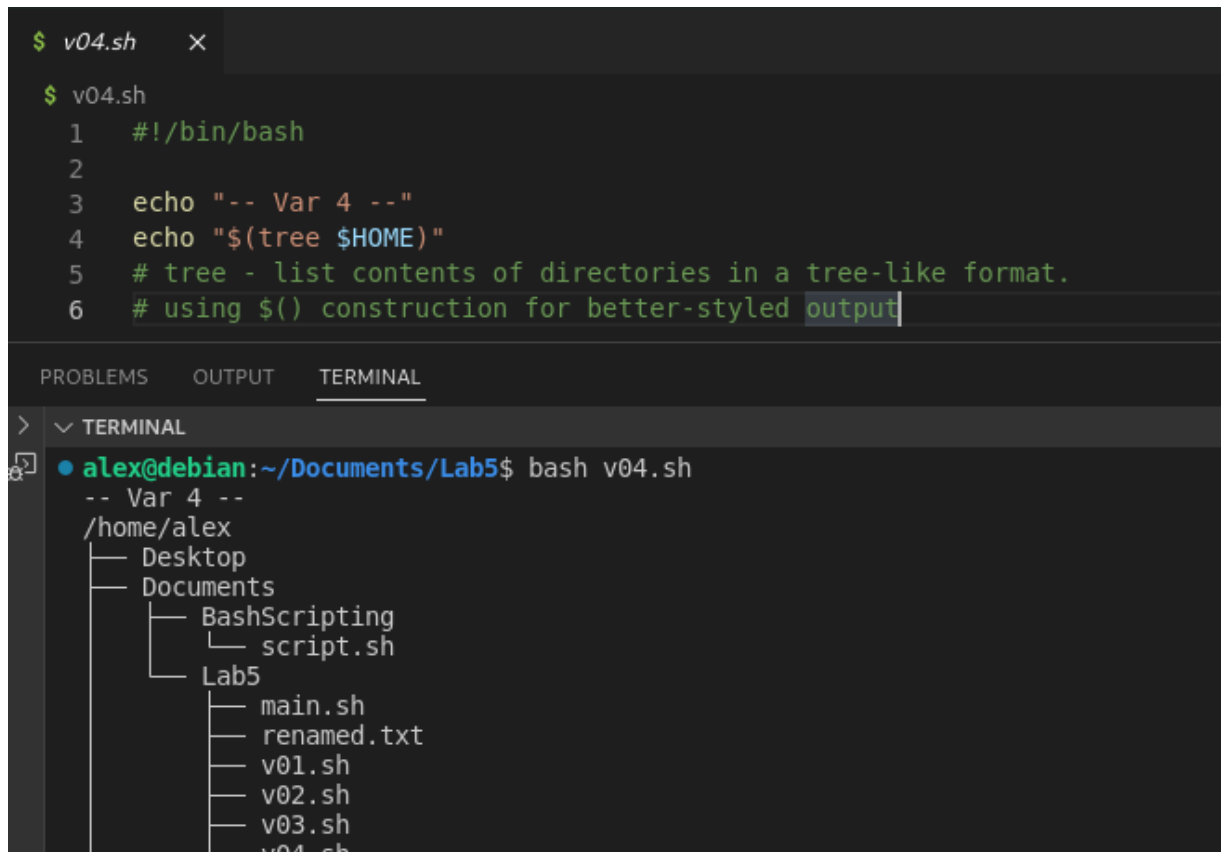
PROBLEMS OUTPUT TERMINAL

> ▾ TERMINAL

```
● alex@debian:~/Documents/Lab5$ bash v03.sh
-- Var 3 --
  PID TTY          TIME CMD
  6209 pts/2        00:00:00 bash
  7248 pts/2        00:00:00 bash
  7249 pts/2        00:00:00 ps
○ alex@debian:~/Documents/Lab5$
```

Варіант 4: Вивести дерево вашого домашнього каталогу.

Використовуємо команду tree для виведення дерева. Також, використовуємо конструкцію підстановки значень. Це потрібно для того, щоб результат команди виводився як звичайний текст, це більш імпонує дизайну тексту скрипту. Також, поточний каталог визначається змінною оболонки \$HOME.



```
$ v04.sh
$ v04.sh
1  #!/bin/bash
2
3  echo "-- Var 4 --"
4  echo "$(tree $HOME)"
5  # tree - list contents of directories in a tree-like format.
6  # using $() construction for better-styled output
```

PROBLEMS OUTPUT TERMINAL

> ▾ TERMINAL

```
● alex@debian:~/Documents/Lab5$ bash v04.sh
-- Var 4 --
/home/alex
├── Desktop
├── Documents
│   ├── BashScripting
│   │   └── script.sh
│   └── Lab5
│       ├── main.sh
│       ├── renamed.txt
│       ├── v01.sh
│       ├── v02.sh
│       ├── v03.sh
│       └── v04.sh
```

Варіант 5: Знайти текстовий рядок у вашому файлі.

Оголошуємо змінну з назвою файлу. Перевіряємо, чи існує файл, використовуючи логічний оператор НЕ. Якщо існує, то зчитуємо з клавіатури слово, яке потрібно знайти. Перевіряємо, чи успішна остання операція. Якщо так – виводимо рядок з потрібним словом на екран, якщо ні – повертаємо повідомлення про те, що нічого не знайдено.

```
$ v05.sh
1  #!/bin/bash
2
3  echo "-- Var 5 --"
4
5  FILE="Kovalyov.txt"
6
7  if [[ !(-f $FILE) ]]; then          # if file doesn't exist
8      echo "File has not found."
9      exit 1
10 fi
11
12 # command "read" reading input from standard input to var.
13 # -p key allows to print text input prompt
14 # grep is used here for finding a string
15 # -i key ignores cases of letters
16 read -p "Input string that you want to find: " STRING
17 LINE=$(grep -i $STRING $FILE)
18
19 # $? - code result of previous operation
20 if [ $? -eq 0 ]; then
21     echo "String found! Result: "
22     echo "$LINE"
23 else
24     echo "String wasn't found."
25 fi
```

PROBLEMS OUTPUT TERMINAL

> TERMINAL

```
● alex@debian:~/Documents/Lab5$ bash v05.sh
-- Var 5 --
Input string that you want to find: tp-12
String found! Result:
Academic group = TP-12
● alex@debian:~/Documents/Lab5$
```

Варіант 6: Вивести інформацію про систему.

Для виведення повної інформації викликаємо утиліту neofetch.

```
$ v06.sh X
$ v06.sh
1 #!/bin/bash
2
3 echo "-- Var 6 --"
4 # neofetch - A fast, highly customizable system info script
5 neofetch
```

PROBLEMS OUTPUT TERMINAL

▼ TERMINAL

```
● alex@debian:~/Documents/Lab5$ bash v06.sh
-- Var 6 --

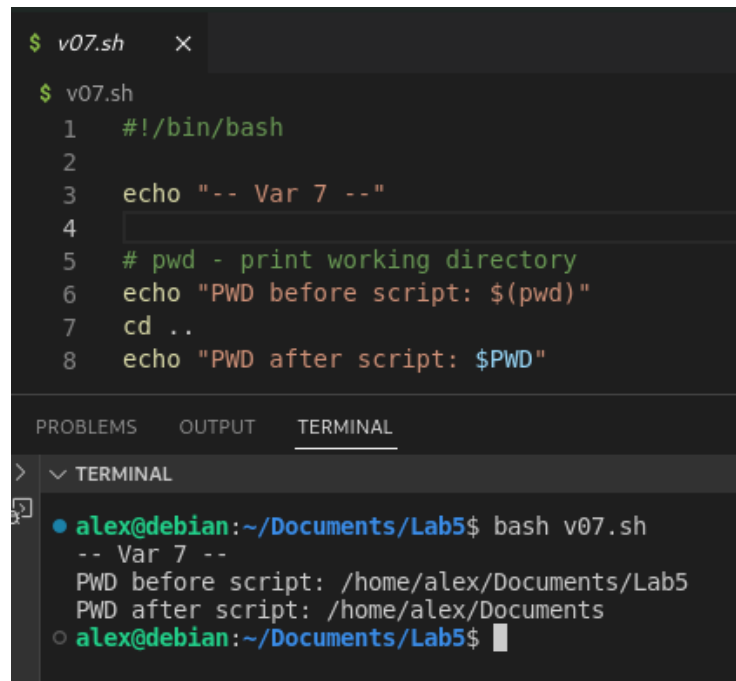
      ,metssssggg.
    ,gssssssssssssssP.
   ,gSSP"" """"Y$$."
  ,SSP"" """"Y$$."
 ,SSP"" """"Y$$."
dsss' ,ggs. $$$b:
     ,SP"" . $$$
SSP ds' . $$$
SS: $$ - d$d$'
SS; Y$b. ,dsp'
YSS: "Y$$$$$P"
$$$b "-_
YSS
YSS$.
$$$b.
Y$b.
"Y$b.""
```

alex@debian

OS: Debian GNU/Linux 11 (bullseye) x86_64
Host: VirtualBox 1.2
Kernel: 5.10.0-19-amd64
Uptime: 2 hours, 33 mins
Packages: 1970 (dpkg)
Shell: bash 5.1.4
Resolution: preferred
DE: GNOME 3.38.6
WM: Mutter
WM Theme: Adwaita
Theme: Green-Submarine [GTK2/3]
Icons: Adwaita [GTK2/3]
Terminal: vscode
CPU: Intel i3-8130U (2) @ 2.208GHz
GPU: 00:02.0 VMware SVGA II Adapter
Memory: 1338M1B / 1982MiB

Варіант 7: Змінити поточний каталог.

Використовуємо команду `cd` для переходу між каталогами. Для перевірки поточного робочого каталогу використовується `PWD` як команда та як змінна оболонки.

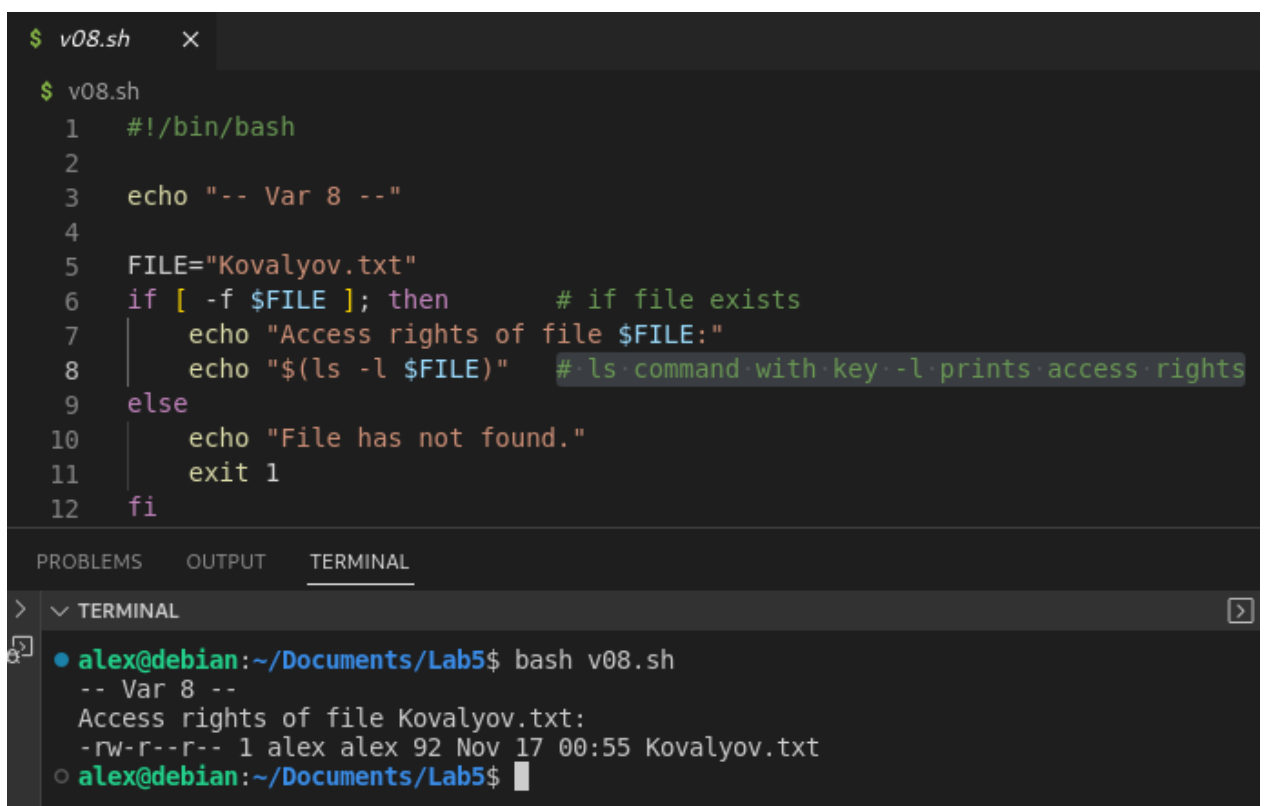


```
$ v07.sh x
$ v07.sh
1  #!/bin/bash
2
3  echo "-- Var 7 --"
4
5  # pwd - print working directory
6  echo "PWD before script: $(pwd)"
7  cd ..
8  echo "PWD after script: $PWD"

PROBLEMS  OUTPUT  TERMINAL
>  TERMINAL
● alex@debian:~/Documents/Lab5$ bash v07.sh
-- Var 7 --
PWD before script: /home/alex/Documents/Lab5
PWD after script: /home/alex/Documents
○ alex@debian:~/Documents/Lab5$
```

Варіант 8: Відобразити режим доступу до вашого файлу.

Виводимо номер варіанту. Записуємо назву файлу в змінну. Перевіряємо, чи існує файл. Якщо так: використовуємо команду `ls` з ключем `-l`, який виводить більш детальну інформацію про файли. Також вказуємо назву файлу, щоб інформація вивелась саме про нього.

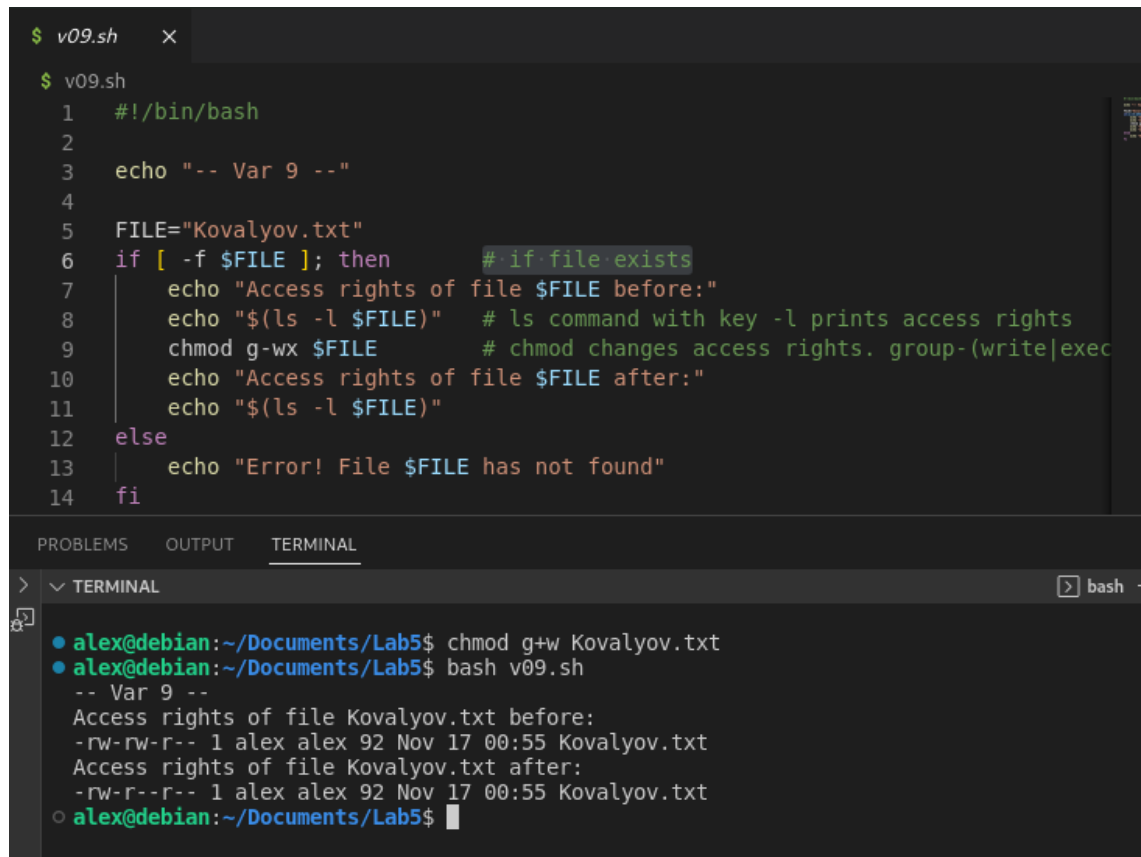


```
$ v08.sh x
$ v08.sh
1  #!/bin/bash
2
3  echo "-- Var 8 --"
4
5  FILE="Kovalyov.txt"
6  if [ -f $FILE ]; then          # if file exists
7      echo "Access rights of file $FILE:"
8      echo "$(ls -l $FILE)"      # ls command with key -l prints access rights
9  else
10     echo "File has not found."
11     exit 1
12 fi

PROBLEMS  OUTPUT  TERMINAL
>  TERMINAL
● alex@debian:~/Documents/Lab5$ bash v08.sh
-- Var 8 --
Access rights of file Kovalyov.txt:
-rw-r--r-- 1 alex alex 92 Nov 17 00:55 Kovalyov.txt
○ alex@debian:~/Documents/Lab5$
```

Варіант 9: Змінити режим доступу до власного файлу групі користувачів для встановлення дозволу тільки на читання.

Перевіряємо, чи існує файл. Виводимо режим доступу до операцій. Перед запуском скрипту змінюємо права, щоб продемонструвати роботу. Забираємо права у групи на читання та виконання за допомогою команди `chmod`.

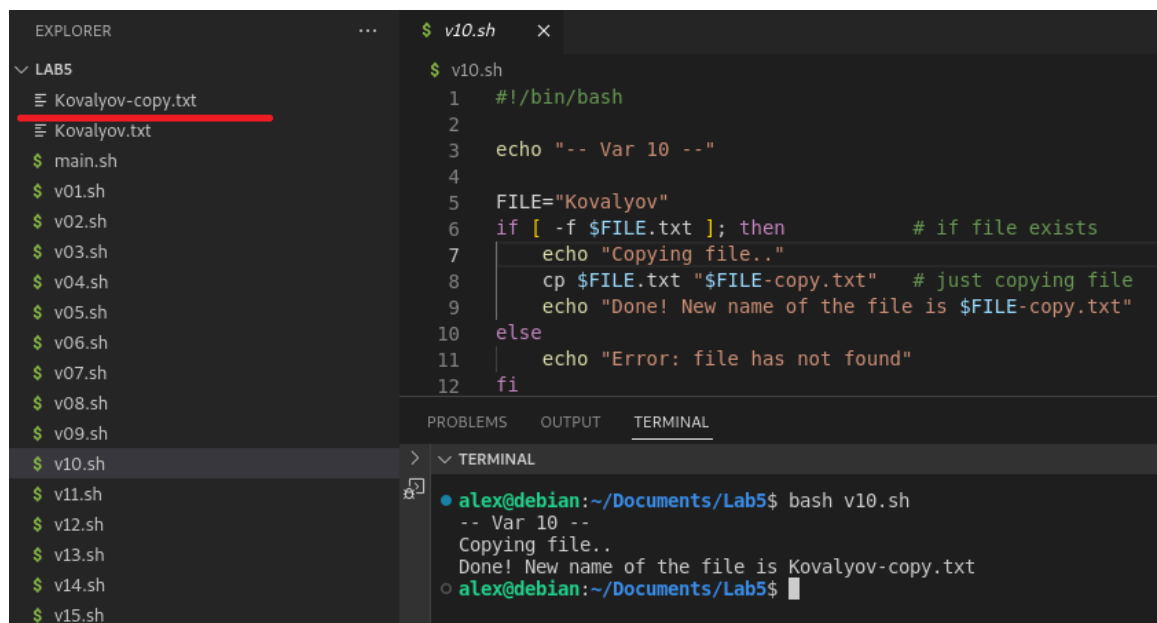


```
$ v09.sh
$ v09.sh
1  #!/bin/bash
2
3  echo "-- Var 9 --"
4
5  FILE="Kovalyov.txt"
6  if [ -f $FILE ]; then          # if file exists
7      echo "Access rights of file $FILE before:"
8      echo "$(ls -l $FILE)"      # ls command with key -l prints access rights
9      chmod g-wx $FILE          # chmod changes access rights. group-(write|exec
10     echo "Access rights of file $FILE after:"
11     echo "$(ls -l $FILE)"
12 else
13     echo "Error! File $FILE has not found"
14 fi

PROBLEMS  OUTPUT  TERMINAL
> v TERMINAL bash
• alex@debian:~/Documents/Lab5$ chmod g+w Kovalyov.txt
• alex@debian:~/Documents/Lab5$ bash v09.sh
-- Var 9 --
Access rights of file Kovalyov.txt before:
-rw-rw-r-- 1 alex alex 92 Nov 17 00:55 Kovalyov.txt
Access rights of file Kovalyov.txt after:
-rw-r--r-- 1 alex alex 92 Nov 17 00:55 Kovalyov.txt
• alex@debian:~/Documents/Lab5$
```

Варіант 10: Виконати копіювання власного файлу зі збереженням атрибутів.

За умовчуванням команда `cp` виконує копіювання зі збереженням атрибутів. Використовуємо її.



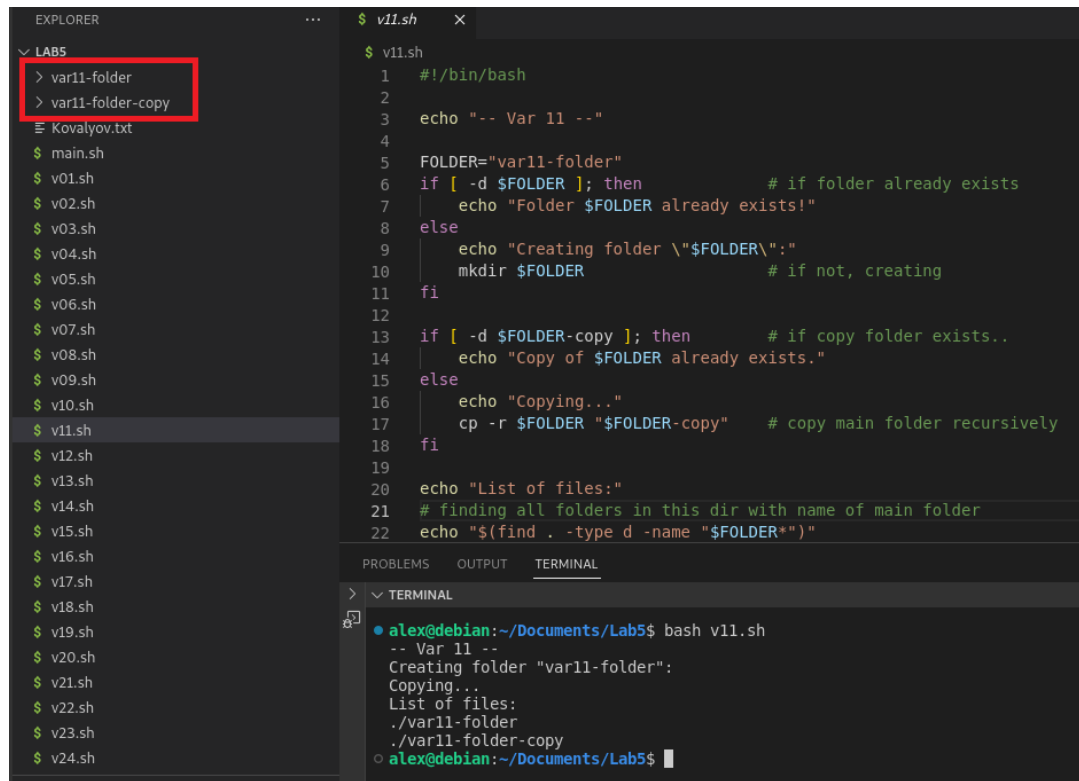
```
EXPLORER
LAB5
  Kovalyov-copy.txt
  Kovalyov.txt
  main.sh
  v01.sh
  v02.sh
  v03.sh
  v04.sh
  v05.sh
  v06.sh
  v07.sh
  v08.sh
  v09.sh
  v10.sh
  v11.sh
  v12.sh
  v13.sh
  v14.sh
  v15.sh

$ v10.sh
$ v10.sh
1  #!/bin/bash
2
3  echo "-- Var 10 --"
4
5  FILE="Kovalyov"
6  if [ -f $FILE.txt ]; then      # if file exists
7      echo "Copying file.."
8      cp $FILE.txt "$FILE-copy.txt" # just copying file
9      echo "Done! New name of the file is $FILE-copy.txt"
10 else
11     echo "Error: file has not found"
12 fi

PROBLEMS  OUTPUT  TERMINAL
> v TERMINAL
• alex@debian:~/Documents/Lab5$ bash v10.sh
-- Var 10 --
Copying file..
Done! New name of the file is Kovalyov-copy.txt
• alex@debian:~/Documents/Lab5$
```

Варіант 11: Створити каталог та виконати його копіювання.

Записуємо у змінну назву каталогу. Про всяк випадок, перевіряємо чи існує ця директорія. Якщо ні – створюємо за допомогою команди `mkdir`. Якщо так – виводимо повідомлення про помилку. Таку ж перевірку проводимо і з можливою копією каталогу. Після цього виводимо список директорій, у яких в назві є назва початкової директорії.



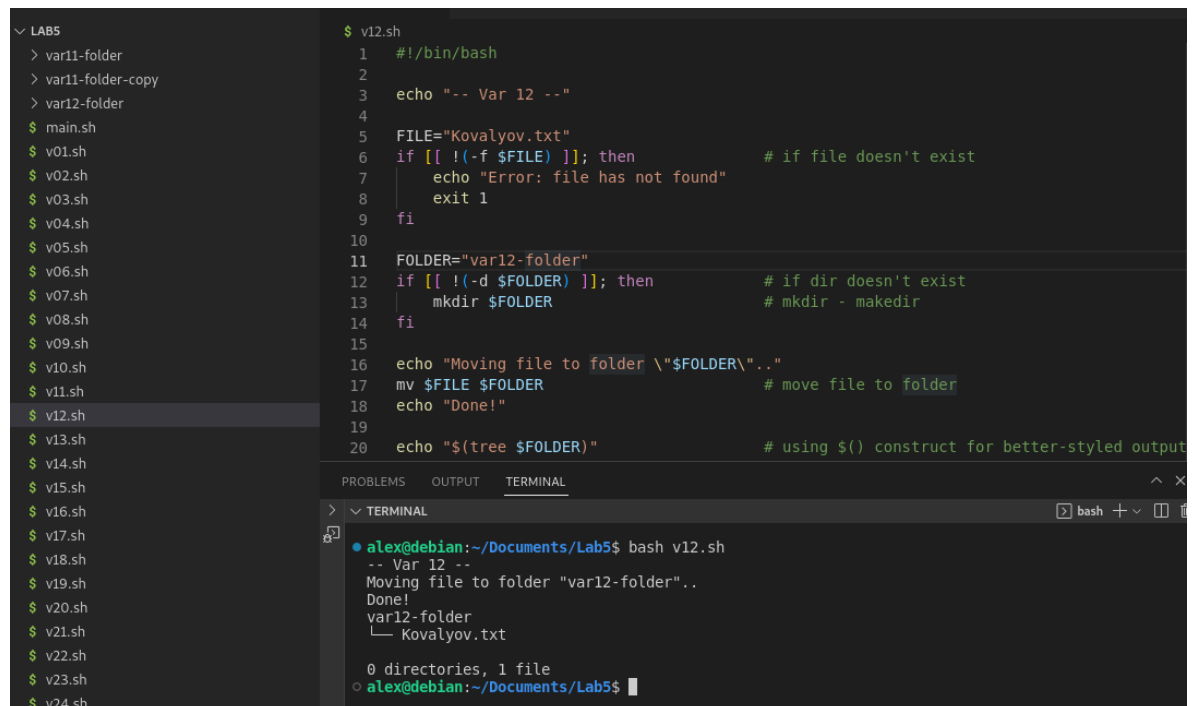
```
EXPLORER
└─ LAB5
  > var11-folder
  > var11-folder-copy
  └─ Kovalyov.txt
    $ main.sh
    $ v01.sh
    $ v02.sh
    $ v03.sh
    $ v04.sh
    $ v05.sh
    $ v06.sh
    $ v07.sh
    $ v08.sh
    $ v09.sh
    $ v10.sh
    $ v11.sh
    $ v12.sh
    $ v13.sh
    $ v14.sh
    $ v15.sh
    $ v16.sh
    $ v17.sh
    $ v18.sh
    $ v19.sh
    $ v20.sh
    $ v21.sh
    $ v22.sh
    $ v23.sh
    $ v24.sh

$ v11.sh
1  #!/bin/bash
2
3  echo "-- Var 11 --"
4
5  FOLDER="var11-folder"
6  if [ -d $FOLDER ]; then          # if folder already exists
7      echo "Folder $FOLDER already exists!"
8  else
9      echo "Creating folder \"$FOLDER\":"
10     mkdir $FOLDER                # if not, creating
11 fi
12
13 if [ -d $FOLDER-copy ]; then    # if copy folder exists..
14     echo "Copy of $FOLDER already exists."
15 else
16     echo "Copying..."
17     cp -r $FOLDER "$FOLDER-copy" # copy main folder recursively
18 fi
19
20 echo "List of files:"
21 # finding all folders in this dir with name of main folder
22 echo "$(find . -type d -name "$FOLDER*")"
```

```
alex@debian:~/Documents/Lab5$ bash v11.sh
-- Var 11 --
Creating folder "var11-folder":
Copying...
List of files:
./var11-folder
./var11-folder-copy
alex@debian:~/Documents/Lab5$
```

Варіант 12: Перемістити власний файл у новий каталог.

Використовуємо перевірку на існування файлу з попередніх завдань. Створюємо новий каталог, та переміщуємо туди власний файл за допомогою команди `mv`.



```
EXPLORER
└─ LAB5
  > var11-folder
  > var11-folder-copy
  > var12-folder
  └─ Kovalyov.txt
    $ main.sh
    $ v01.sh
    $ v02.sh
    $ v03.sh
    $ v04.sh
    $ v05.sh
    $ v06.sh
    $ v07.sh
    $ v08.sh
    $ v09.sh
    $ v10.sh
    $ v11.sh
    $ v12.sh
    $ v13.sh
    $ v14.sh
    $ v15.sh
    $ v16.sh
    $ v17.sh
    $ v18.sh
    $ v19.sh
    $ v20.sh
    $ v21.sh
    $ v22.sh
    $ v23.sh
    $ v24.sh

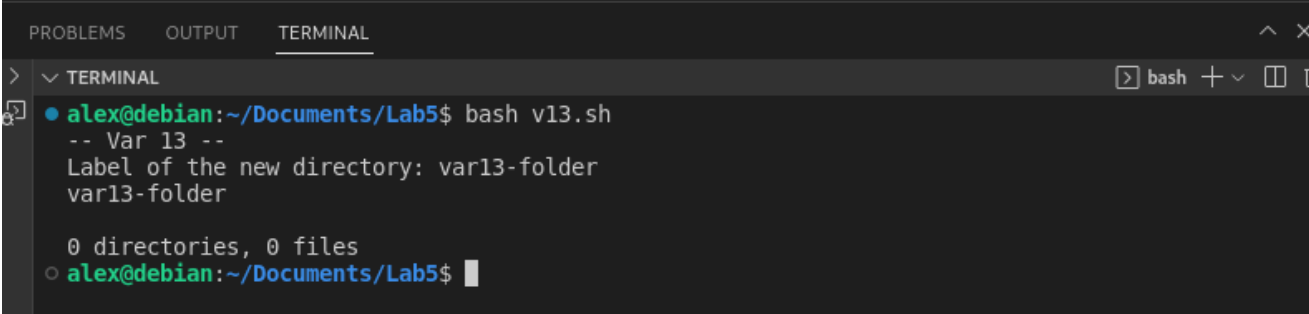
$ v12.sh
1  #!/bin/bash
2
3  echo "-- Var 12 --"
4
5  FILE="Kovalyov.txt"
6  if [ !(-f $FILE) ]; then        # if file doesn't exist
7      echo "Error: file has not found"
8      exit 1
9  fi
10
11 FOLDER="var12-folder"
12 if [ !(-d $FOLDER) ]; then      # if dir doesn't exist
13     mkdir $FOLDER               # mkdir - makedir
14 fi
15
16 echo "Moving file to folder \"$FOLDER\".."
17 mv $FILE $FOLDER                # move file to folder
18 echo "Done!"
19
20 echo "$(tree $FOLDER)"          # using $( ) construct for better-styled output
```

```
alex@debian:~/Documents/Lab5$ bash v12.sh
-- Var 12 --
Moving file to folder "var12-folder"..
Done!
var12-folder
└─ Kovalyov.txt
0 directories, 1 file
alex@debian:~/Documents/Lab5$
```

Варіант 13: Створити каталог, вивести його назву на екран та видалити.

Записуємо назву каталогу у змінну. Якщо такого не існує, то створюємо новий. Виводимо назву на екран. Демонструємо дерево каталогу як доказ того, що він пустий. Використовуємо команду `rmdir`, так як впевнені, що там пусто.

```
$ v13.sh
1  #!/bin/bash
2
3  echo "-- Var 13 --"
4
5  FOLDER="var13-folder"
6
7  if [ [ !(-d $FOLDER) ] ]; then          # if dir doesn't exist
8  |   mkdir $FOLDER                        # mkdir - makedir
9  fi
10
11 echo "Label of the new directory: $FOLDER"
12 echo "$(tree $FOLDER)"                  # using $() construct for better-styled output
13 rmdir $FOLDER                           # if it's clear
```



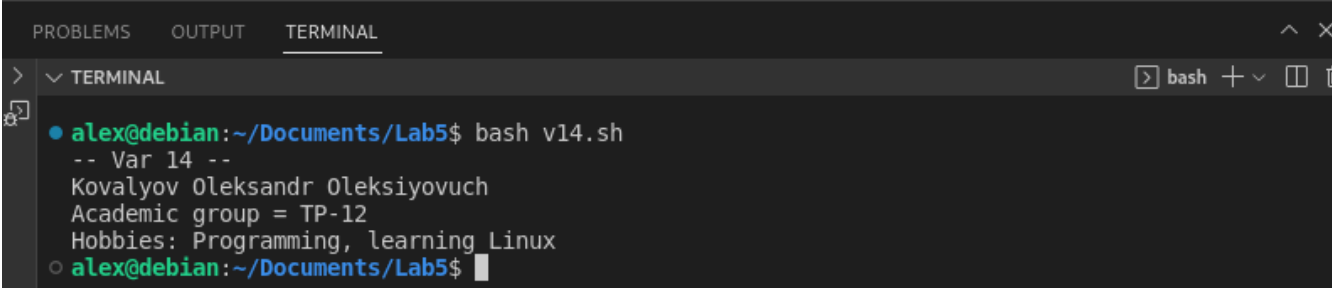
```
PROBLEMS  OUTPUT  TERMINAL
>  ▾ TERMINAL
● alex@debian:~/Documents/Lab5$ bash v13.sh
-- Var 13 --
Label of the new directory: var13-folder
var13-folder

0 directories, 0 files
○ alex@debian:~/Documents/Lab5$
```

Варіант 14: Вивести на екран зміст вашого файлу.

Перевіряємо чи існує файл, якщо так, то виводимо на екран зміст за допомогою команди `cat`. Інакше – виводимо повідомлення про помилку.

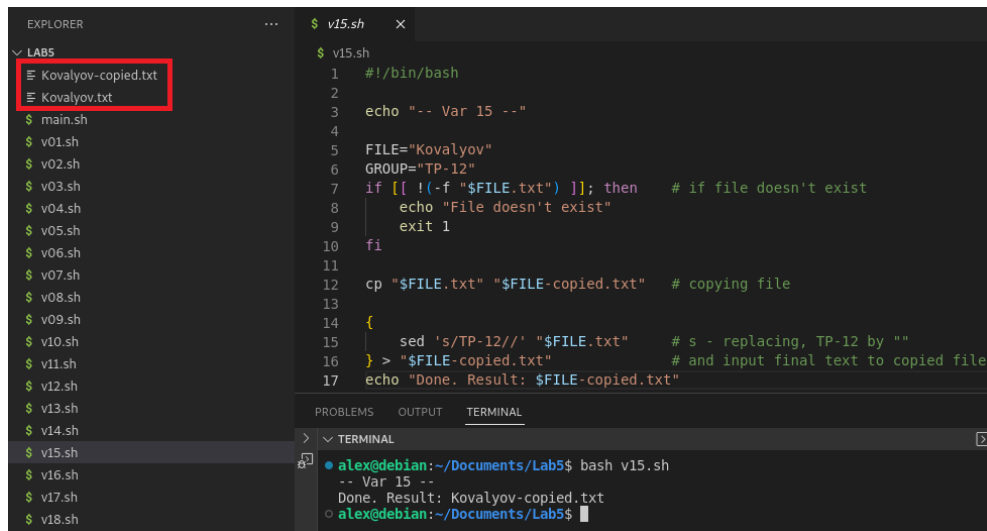
```
$ v14.sh
1  #!/bin/bash
2
3  echo "-- Var 14 --"
4
5  FILE="Kovalyov.txt"
6  if [ -f $FILE ]; then                  # if file exists
7  |   cat $FILE                          # cat - concatenate files and print on the standard ou
8  else
9  |   echo "File doesn't exist"
10 fi
```



```
PROBLEMS  OUTPUT  TERMINAL
>  ▾ TERMINAL
● alex@debian:~/Documents/Lab5$ bash v14.sh
-- Var 14 --
Kovalyov Oleksandr Oleksiyovuch
Academic group = TP-12
Hobbies: Programming, learning Linux
○ alex@debian:~/Documents/Lab5$
```

Варіант 15: Скопіювати власний файл, видалити у ньому назву групи.

Записуємо у змінну назву файлу та групи. Перевіряємо, чи існує. Якщо так, копіюємо. В анонімній функції використовуємо утиліту sed, яка редагує текст. Видаляємо назву групи, записуємо все у скопійований файл. Таке рішення є швидшим ніж перебір кожного рядку циклом.



```
EXPLORER
└─ LAB5
   ├── Kovalyov-copied.txt
   └── Kovalyov.txt

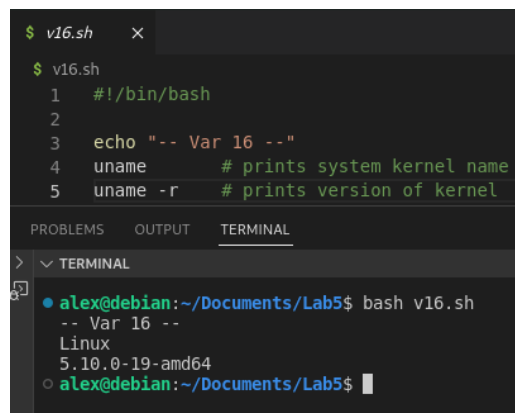
$ main.sh
$ v01.sh
$ v02.sh
$ v03.sh
$ v04.sh
$ v05.sh
$ v06.sh
$ v07.sh
$ v08.sh
$ v09.sh
$ v10.sh
$ v11.sh
$ v12.sh
$ v13.sh
$ v14.sh
$ v15.sh
$ v16.sh
$ v17.sh
$ v18.sh

$ v15.sh
1  #!/bin/bash
2
3  echo "-- Var 15 --"
4
5  FILE="Kovalyov"
6  GROUP="TP-12"
7  if [ !(-f "$FILE.txt") ]; then      # if file doesn't exist
8      echo "File doesn't exist"
9      exit 1
10 fi
11
12 cp "$FILE.txt" "$FILE-copied.txt"  # copying file
13
14 {
15     sed 's/TP-12//' "$FILE.txt"    # s - replacing, TP-12 by ""
16 } > "$FILE-copied.txt"            # and input final text to copied file
17 echo "Done. Result: $FILE-copied.txt"

PROBLEMS  OUTPUT  TERMINAL
└─ TERMINAL
   ● alex@debian:~/Documents/Lab5$ bash v15.sh
     -- Var 15 --
     Done. Result: Kovalyov-copied.txt
   ○ alex@debian:~/Documents/Lab5$
```

Варіант 16: Вивести версію системи.

Для виведення назви ядра та версії системи використовуємо команду uname.

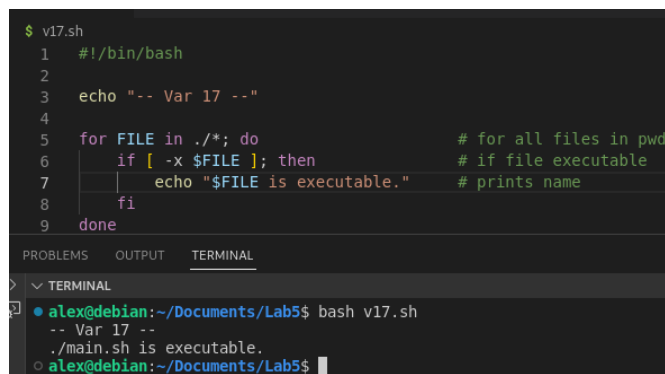


```
$ v16.sh
$ v16.sh
1  #!/bin/bash
2
3  echo "-- Var 16 --"
4  uname      # prints system kernel name
5  uname -r   # prints version of kernel

PROBLEMS  OUTPUT  TERMINAL
└─ TERMINAL
   ● alex@debian:~/Documents/Lab5$ bash v16.sh
     -- Var 16 --
     Linux
     5.10.0-19-amd64
   ○ alex@debian:~/Documents/Lab5$
```

Варіант 17: В каталозі віднайти виконуваний файл.

За допомогою циклу перебираємо всі файли у каталозі. Якщо файл виконуваний (ключ -x, executable) то виводимо на екран.

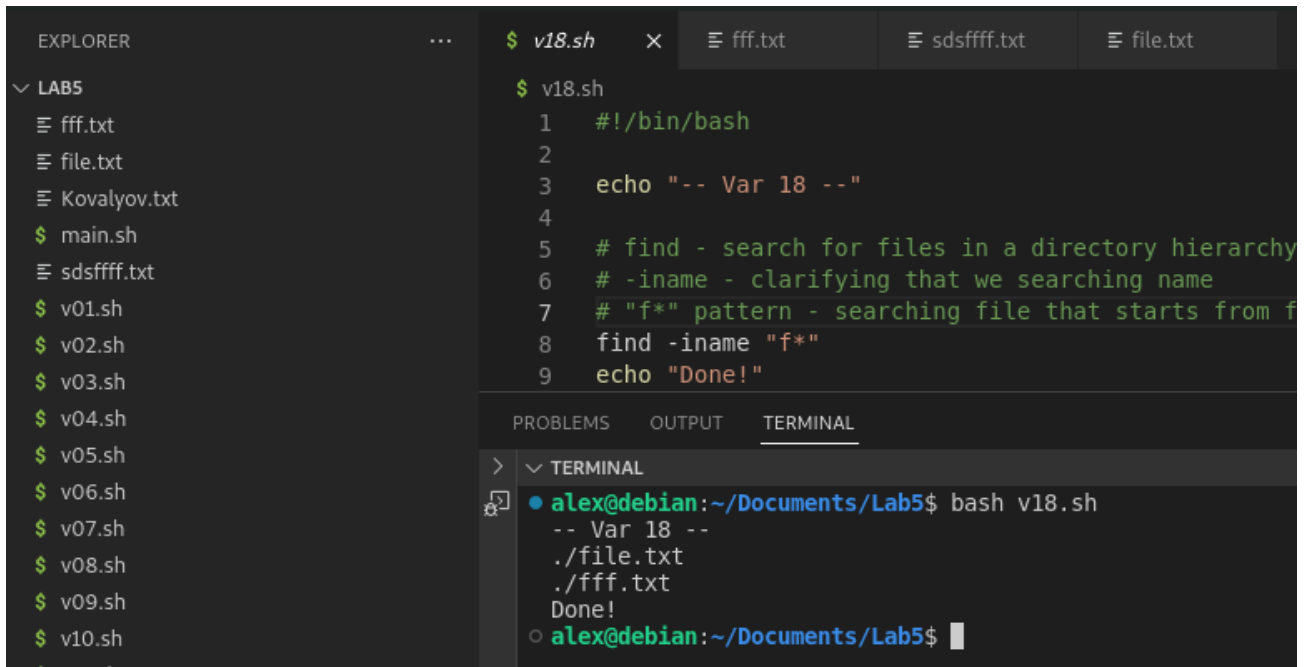


```
$ v17.sh
1  #!/bin/bash
2
3  echo "-- Var 17 --"
4
5  for FILE in ./*; do                # for all files in pwd
6      if [ -x $FILE ]; then          # if file executable
7          echo "$FILE is executable." # prints name
8      fi
9  done

PROBLEMS  OUTPUT  TERMINAL
└─ TERMINAL
   ● alex@debian:~/Documents/Lab5$ bash v17.sh
     -- Var 17 --
     ./main.sh is executable.
   ○ alex@debian:~/Documents/Lab5$
```


Варіант 18: Вивести імена файлів, які починаються на букву «f».

Команда `find` шукає файли у заданому каталозі. Каталог не вказаний, бо пошук проводиться у поточному. Ключ `-iname` означає, що пошук відбувається за іменем (наступний аргумент) та ігноруючи регістр. Вказуємо регулярний вираз.



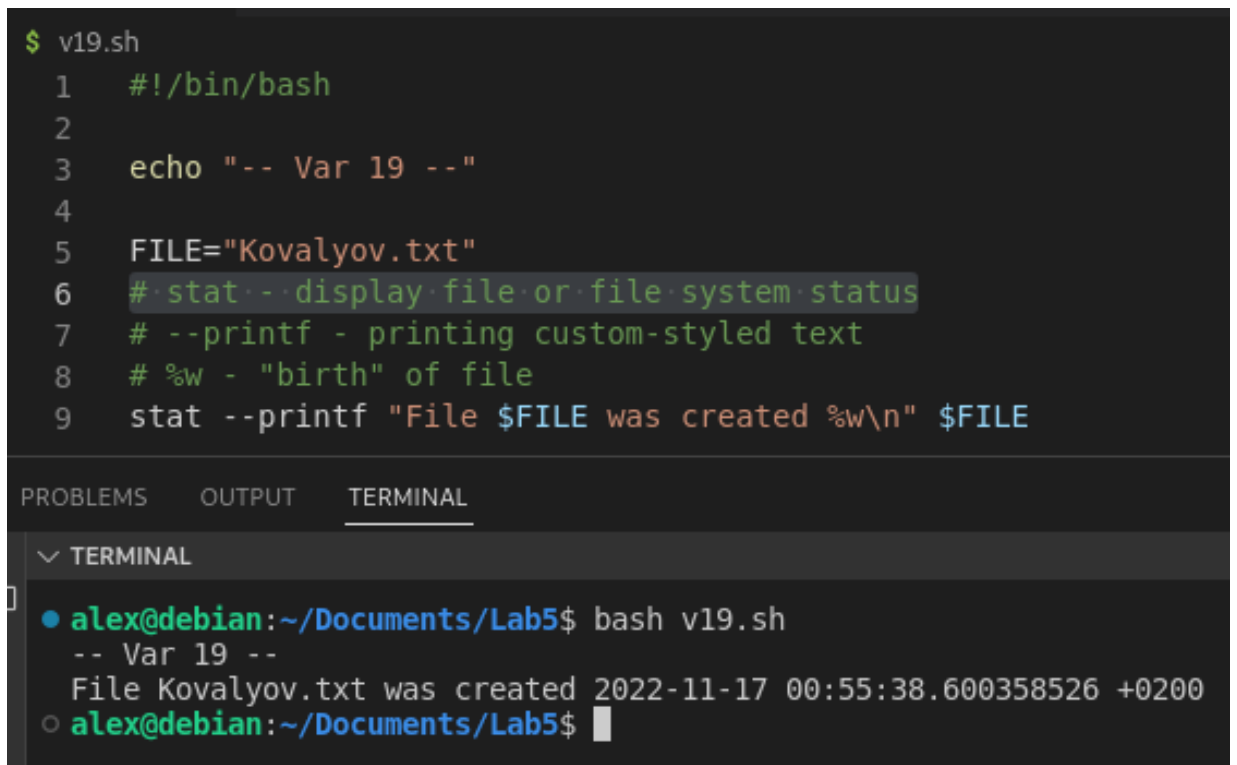
```
EXPLORER
  LAB5
    fff.txt
    file.txt
    Kovalyov.txt
    $ main.sh
    sdsffff.txt
    $ v01.sh
    $ v02.sh
    $ v03.sh
    $ v04.sh
    $ v05.sh
    $ v06.sh
    $ v07.sh
    $ v08.sh
    $ v09.sh
    $ v10.sh

$ v18.sh
1  #!/bin/bash
2
3  echo "-- Var 18 --"
4
5  # find - search for files in a directory hierarchy
6  # -iname - clarifying that we searching name
7  # "f*" pattern - searching file that starts from f
8  find -iname "f*"
9  echo "Done!"

PROBLEMS  OUTPUT  TERMINAL
  >
  v TERMINAL
  ● alex@debian:~/Documents/Lab5$ bash v18.sh
    -- Var 18 --
    ./file.txt
    ./fff.txt
    Done!
  ○ alex@debian:~/Documents/Lab5$
```

Варіант 19: Вивести на екран дату створення файлу (власного або будь-якого іншого).

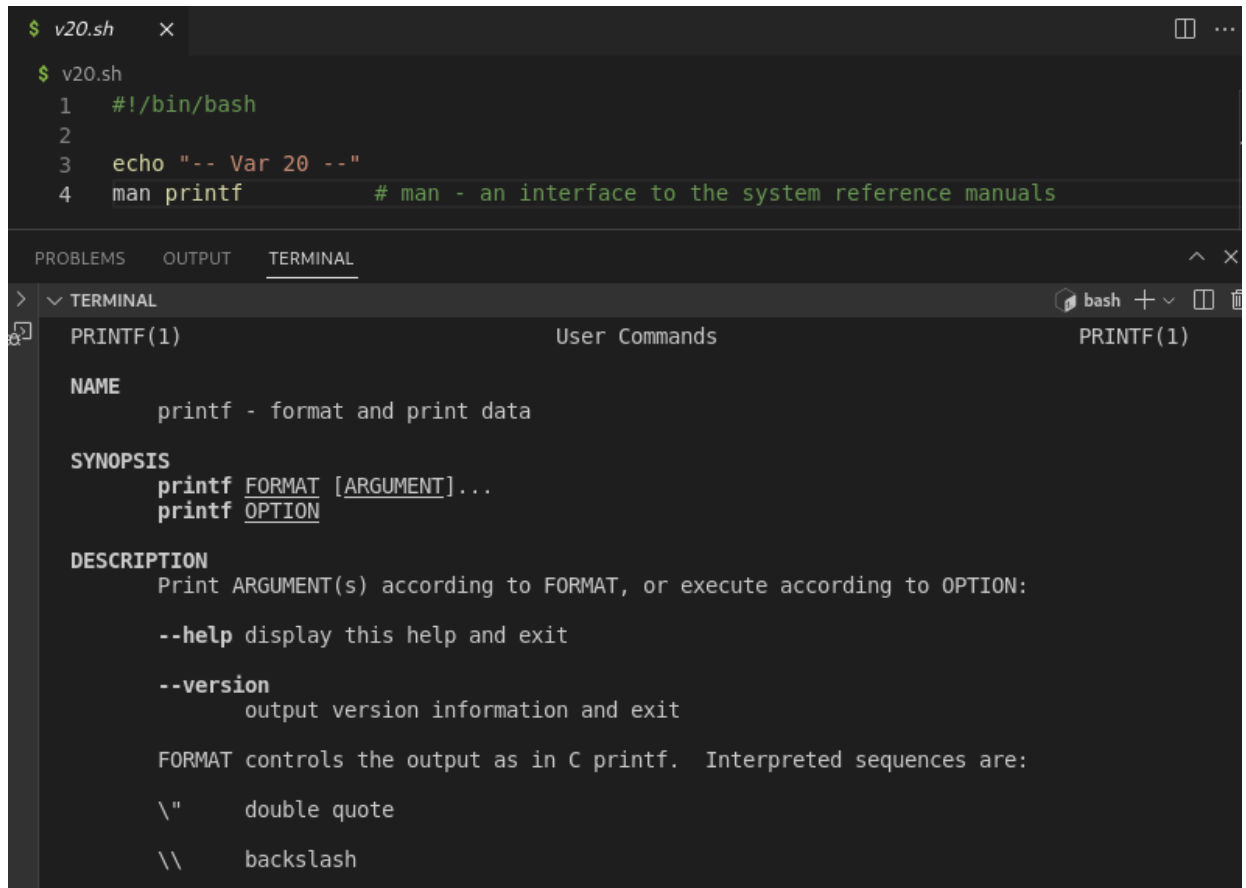
Команда `stat` виводить інформацію про файл. Ключ `--printf` дозволяє відредагувати кінцеве повідомлення. Специфікатор формату `%w` означає, що нам потрібна дата. `\n` – керуюча послідовність, перехід на новий рядок.



```
$ v19.sh
1  #!/bin/bash
2
3  echo "-- Var 19 --"
4
5  FILE="Kovalyov.txt"
6  # stat --display file or file system status
7  # --printf - printing custom-styled text
8  # %w - "birth" of file
9  stat --printf "File $FILE was created %w\n" $FILE

PROBLEMS  OUTPUT  TERMINAL
  >
  v TERMINAL
  ● alex@debian:~/Documents/Lab5$ bash v19.sh
    -- Var 19 --
    File Kovalyov.txt was created 2022-11-17 00:55:38.600358526 +0200
  ○ alex@debian:~/Documents/Lab5$
```

Варіант 20: Вивести довідку про команду printf.
Команда man (від англ. Manual) виводить довідку.



```
$ v20.sh
$ v20.sh
1  #!/bin/bash
2
3  echo "-- Var 20 --"
4  man printf          # man - an interface to the system reference manuals
```

PROBLEMS OUTPUT **TERMINAL**

PRINTF(1) User Commands PRINTF(1)

NAME

printf - format and print data

SYNOPSIS

printf **FORMAT** [**ARGUMENT**]...

printf **OPTION**

DESCRIPTION

Print **ARGUMENT**(s) according to **FORMAT**, or execute according to **OPTION**:

--help display this help and exit

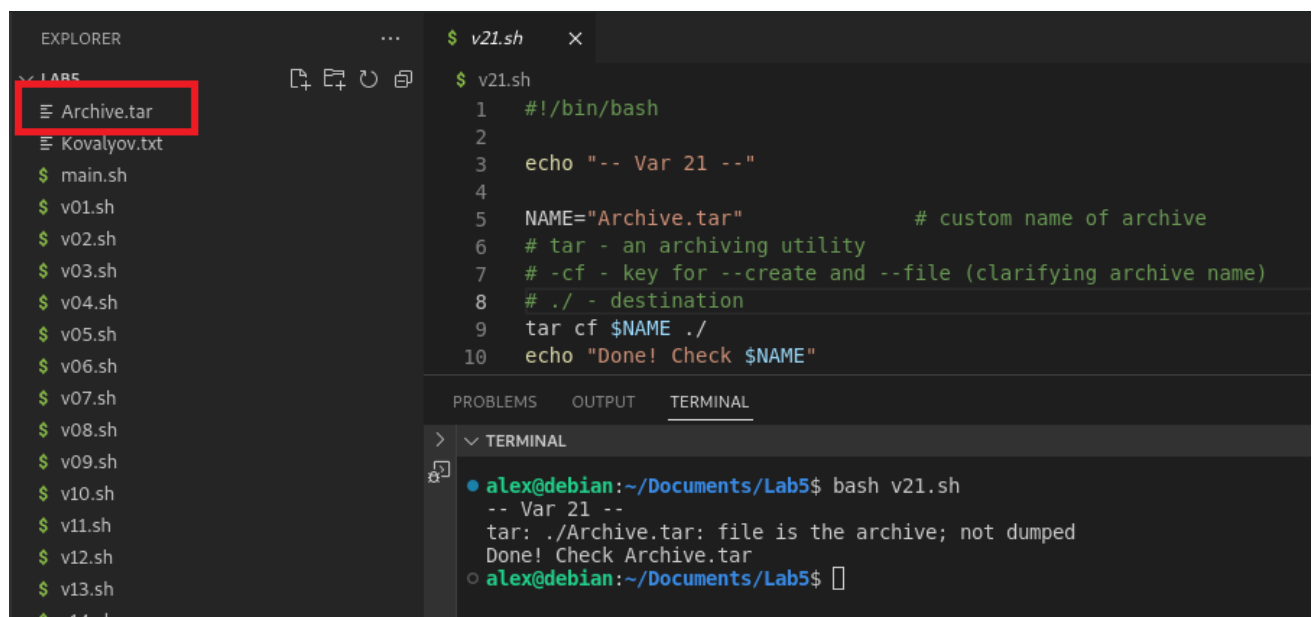
--version output version information and exit

FORMAT controls the output as in C printf. Interpreted sequences are:

\ " double quote

\\ backslash

Варіант 21: Створити архів для вашого файлу або каталогу, де він зберігається.
Записуємо у змінну назву архіву. Використовуємо утиліту tar – вона дозволяє архівувати файли. Ключ -cf означає create file – тобто створити архів зі вказаним ім'ям. Вказуємо поточний каталог. Виводимо назву архіву. Перевіряємо результат – серед файлів видно архів.



```
$ v21.sh
$ v21.sh
1  #!/bin/bash
2
3  echo "-- Var 21 --"
4
5  NAME="Archive.tar"          # custom name of archive
6  # tar - an archiving utility
7  # -cf - key for --create and --file (clarifying archive name)
8  # ./ - destination
9  tar cf $NAME ./
10 echo "Done! Check $NAME"
```

PROBLEMS OUTPUT **TERMINAL**

EXPLORER

Archive.tar

Kovalyov.txt

\$ main.sh

\$ v01.sh

\$ v02.sh

\$ v03.sh

\$ v04.sh

\$ v05.sh

\$ v06.sh

\$ v07.sh

\$ v08.sh

\$ v09.sh

\$ v10.sh

\$ v11.sh

\$ v12.sh

\$ v13.sh

\$ v14.sh

alex@debian:~/Documents/Lab5\$ bash v21.sh

-- Var 21 --

tar: ./Archive.tar: file is the archive; not dumped

Done! Check Archive.tar

alex@debian:~/Documents/Lab5\$

Варіант 22: Вивести список інстальованих програм менеджера пакетів dpkg.

Утиліта dpkg – низькорівневий пакетний менеджер. З ключем -l виводить всі інстальовані пакети.

```
$ v22.sh
$ v22.sh
1  #!/bin/bash
2
3  echo "-- Var 22 --"
4  # dpkg - package manager for Debian
5  # -l - list of installed packages
6  dpkg -l
```

```
alex@debian:~/Documents/Lab5$ bash v22.sh
-- Var 22 --
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name                               Version                               Architecture Desc
+++-----+-----+-----+-----+
ii accountsservice                     0.6.55-3                             amd64        quer
ii acl                                 2.2.53-10                            amd64        acce
ii adduser                             3.118                                all          add
ii adwaita-icon-theme                  3.38.0-1                             all          defa
ii aisleriot                           1:3.22.9-1                           amd64        GNOM
ii alsa-topology-conf                  1.2.4-1                              all          ALSA
ii alsa-ucm-conf                       1.2.4-2                              all          ALSA
ii alsa-utils                          1.2.4-1                              amd64        Util
ii anacron                            2.3-30                               amd64        cron
ii apache2-bin                         2.4.54-1~deb11u1                     amd64        Apac
ii apg                                2.2.3.dfsg.1-5+b2                    amd64        Auto
ii apparmor                            2.13.6-10                            amd64        user
```

Варіант 23: Вивести довідку про менеджер пакету aptitude.

Aptitude – високорівневий менеджер пакетів. Man – команда, яка виводить довідку.

```
$ v23.sh
$ v23.sh
1  #!/bin/bash
2
3  echo "-- Var 23 --"
4  # aptitude - high-level interface to the package manager
5  # man - an interface to the system reference manuals
6  man aptitude
```

```
APTITUDE(8)                                Command-line reference                                APTITUDE(8)

NAME
    aptitude - high-level interface to the package manager

SYNOPSIS
    aptitude [<options>...] {autoclean | clean | forget-new | keep-all | update}

    aptitude [<options>...] {full-upgrade | safe-upgrade} [<packages>...]

    aptitude [<options>...] {build-dep | build-depends | changelog | download |
    forbid-version | hold | install | markauto | purge | reinstall | remove |
    show | showsrc | source | unhold | unmarkauto | versions} [<packages>...]

    aptitude extract-cache-subset <output-directory> [<packages>...]

    aptitude [<options>...] search <patterns>...
```

Варіант 24: Відобразити стан поточної конфігурації мережі.

Перевіряємо за допомогою низькорівневого менеджера пакетів `dpkg` чи встановлений пакет `net-tools`. Також застосовуємо утиліту `grep`, яка перевіряє статус у поверненому повідомленні. Якщо все знайшлось – то, відповідно, пакет встановлений. Якщо ні, то за допомогою менеджера пакетів `apt` встановлюємо пакет (потрібні права `sudo`). Після цього запускаємо команду `ifconfig`, яка виводить поточний стан мережі.

```
$ v24.sh
$ v24.sh
1  #!/bin/bash
2
3  echo "-- Var 24 --"
4  # you can check net status by tool ifconfig from net-tools package
5  # is net-tools installed?
6  if [[ !$(dpkg -s net-tools | grep Status) ]]; then
7      # if not, install package by apt
8      sudo apt install net-tools
9  fi
10 # using command
11 sudo ifconfig
```

PROBLEMS OUTPUT TERMINAL

▼ TERMINAL

```
alex@debian:~/Documents/Lab5$ bash v24.sh
-- Var 24 --
[sudo] password for alex:
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255
    inet6 fe80::a00:27ff:fe40:9686 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:40:96:86 txqueuelen 1000 (Ethernet)
    RX packets 2600 bytes 1896611 (1.8 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1611 bytes 1501319 (1.4 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 38 bytes 3523 (3.4 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 38 bytes 3523 (3.4 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Варіант 25: Вивести інформацію про поточного користувача.

Поле `GECOS` має найбільше інформації про користувача. Тому дивимось у файл `/etc/passwd`. Утиліта `cat` виводить передає у потік вміст файлу. Через конвейєр передаємо все утиліті `grep`, яка шукає (ігноруючи регістр, `-i`) користувача (команда `whoami`). Виводимо інформацію на екран.

```
$ v25.sh
$ v25.sh
1  #!/bin/bash
2
3  echo "-- Var 25 --"
4  # cat - concatenate files and print on the standard output
5  # /etc/passwd - file with user details
6  # whoami - prints user name
7  # grep -i: searching user in passwd (ignoring case)
8  cat /etc/passwd | grep -i $(whoami)
```

PROBLEMS OUTPUT TERMINAL

▼ TERMINAL

```
alex@debian:~/Documents/Lab5$ bash v25.sh
-- Var 25 --
alex:x:1000:1000:Alex Kovalyov:/home/alex:/bin/bash
alex@debian:~/Documents/Lab5$
```

Варіант 26: Вивести інформацію про поточний каталог.

Команда `stat` виводить інформацію про файл або директорию. `$PWD` – змінна оболонки, містить в собі поточний каталог. Виводимо інформацію про поточний каталог.

```
$ v26.sh x
$ v26.sh
1  #!/bin/bash
2
3  echo "-- Var 26 --"
4  # stat - display file or file system status
5  # PWD - print working directory
6  stat $PWD
```

PROBLEMS OUTPUT TERMINAL

TERMINAL

```
alex@debian:~/Documents/Lab5$ bash v26.sh
-- Var 26 --
File: /home/alex/Documents/Lab5
Size: 4096      Blocks: 8      IO Block: 4096  directory
Device: 801h/2049d Inode: 914600 Links: 2
Access: (0755/drwxr-xr-x)  Uid: ( 1000/   alex)  Gid: ( 1000/   alex)
Access: 2022-11-17 12:26:41.910235582 +0200
Modify: 2022-11-17 12:24:21.956294402 +0200
Change: 2022-11-17 12:24:21.956294402 +0200
Birth: 2022-11-14 00:48:06.849953416 +0200
alex@debian:~/Documents/Lab5$
```

Варіант 27: Створити новий файл, записати до нього вміст вашого файлу.

Перевіряємо, чи файл існує. Зчитуємо назву нового файлу. Створюємо його за допомогою команди `touch`. Команда `cat` дозволяє перенаправити вміст файлу за допомогою конвейерів. Таким чином записуємо вміст власного файлу у новий, скопійований.

EXPLORER

LAB5

- copied.txt
- Kovalyov.txt
- main.sh
- v01.sh
- v02.sh
- v03.sh
- v04.sh
- v05.sh
- v06.sh
- v07.sh
- v08.sh
- v09.sh
- v10.sh
- v11.sh
- v12.sh
- v13.sh
- v14.sh
- v15.sh
- v16.sh
- v17.sh
- v18.sh
- v19.sh
- v20.sh
- v21.sh
- v22.sh
- v23.sh
- v24.sh

```
$ v27.sh x
$ v27.sh
1  #!/bin/bash
2
3  echo "-- Var 27 --"
4
5  FILE="Kovalyov"
6
7  # is file exist?
8  if [ !(-f "$FILE.txt") ]; then
9      echo "File has not found."
10     exit 1
11 fi
12
13 # command "read" reading input from standard input to var.
14 # -p key allows to print text input prompt
15 read -p "Input label of copy document: " NEWFILE
16
17 # creating new file
18 touch "$NEWFILE.txt"
19
20 # copying text using pipe
21 cat "$FILE.txt" > "$NEWFILE.txt"
22
23 echo "Done. Wrote into $NEWFILE.txt"
```

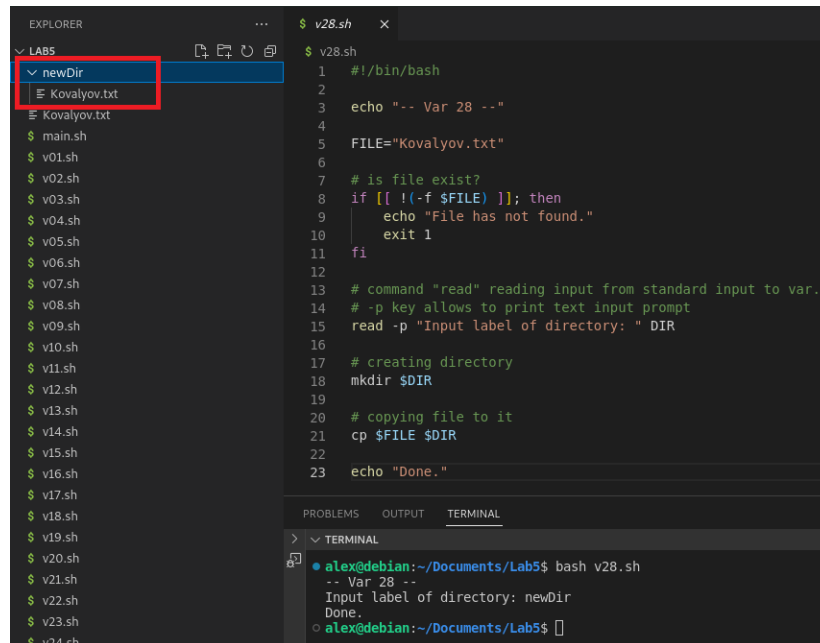
PROBLEMS OUTPUT TERMINAL

TERMINAL

```
alex@debian:~/Documents/Lab5$ bash v27.sh
-- Var 27 --
Input label of copy document: copied
Done. Wrote into copied.txt
alex@debian:~/Documents/Lab5$
```

Варіант 28: Створити новий каталог, скопіювати до нього ваш файл.

Перевіряємо, чи існує початковий файл. Якщо так, зчитуємо назву для нової директорії. Створюємо каталог за допомогою mkdir. Копіюємо файл у директорію використовуючи cp.



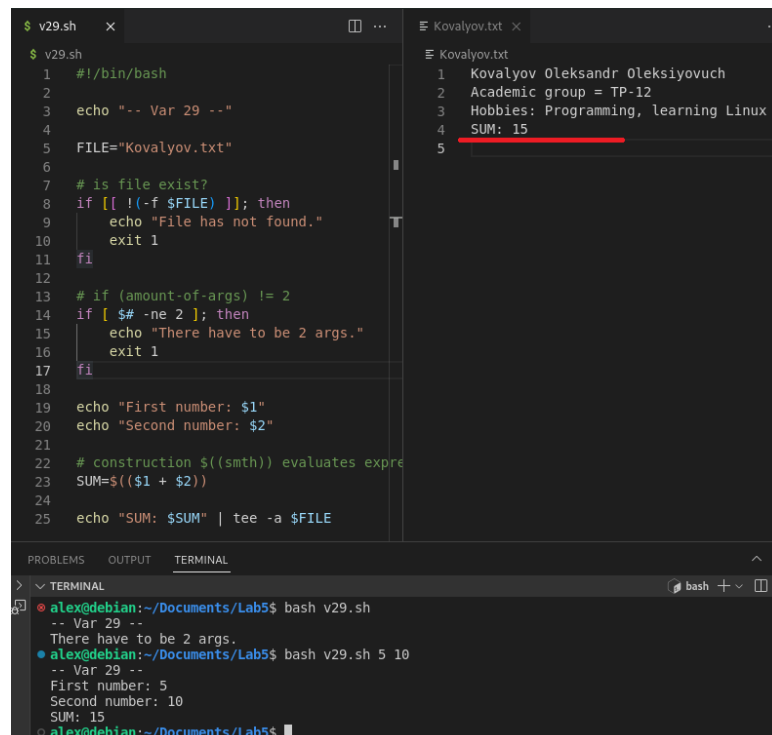
```
EXPLORER
└─ LAB5
   └─ newDir
      └─ Kovalyov.txt
         └─ Kovalyov.txt
            └─ main.sh
               └─ v01.sh
                  └─ v02.sh
                     └─ v03.sh
                        └─ v04.sh
                           └─ v05.sh
                              └─ v06.sh
                                 └─ v07.sh
                                    └─ v08.sh
                                       └─ v09.sh
                                          └─ v10.sh
                                             └─ v11.sh
                                                └─ v12.sh
                                                   └─ v13.sh
                                                      └─ v14.sh
                                                         └─ v15.sh
                                                            └─ v16.sh
                                                               └─ v17.sh
                                                                  └─ v18.sh
                                                                     └─ v19.sh
                                                                        └─ v20.sh
                                                                           └─ v21.sh
                                                                              └─ v22.sh
                                                                                 └─ v23.sh
                                                                                    └─ v24.sh

v28.sh
1  #!/bin/bash
2
3  echo "-- Var 28 --"
4
5  FILE="Kovalyov.txt"
6
7  # is file exist?
8  if [[ !(-f $FILE) ]]; then
9      echo "File has not found."
10     exit 1
11 fi
12
13 # command "read" reading input from standard input to var.
14 # -p key allows to print text input prompt
15 read -p "Input label of directory: " DIR
16
17 # creating directory
18 mkdir $DIR
19
20 # copying file to it
21 cp $FILE $DIR
22
23 echo "Done."
```

```
alex@debian:~/Documents/Lab5$ bash v28.sh
-- Var 28 --
Input label of directory: newDir
Done.
alex@debian:~/Documents/Lab5$
```

Варіант 29: Ввести з командного рядка значення двох змінних, обчислити їх суму, записати у ваш файл.

Перевіряємо, чи існує файл. Якщо так, робимо перевірку на кількість аргументів – їх повинно бути 2. Виводимо числа. Використовуємо конструкцію «подвійні круглі дужки» для обрахування арифметичного виразу. Результат передаємо конвейєром в утиліту tee, яка з ключем -a доповнює файл та виводить все на екран.



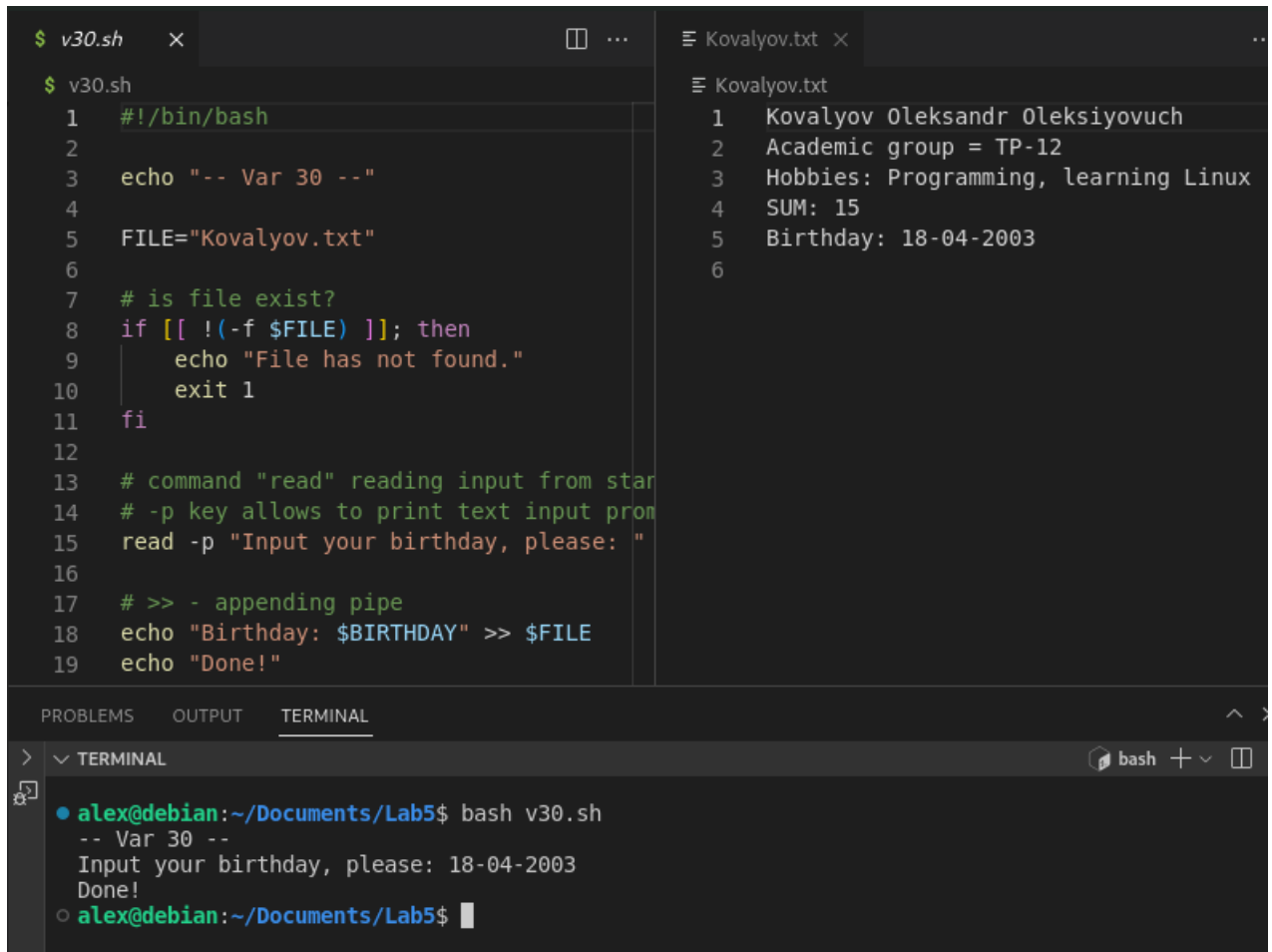
```
v29.sh
1  #!/bin/bash
2
3  echo "-- Var 29 --"
4
5  FILE="Kovalyov.txt"
6
7  # is file exist?
8  if [[ !(-f $FILE) ]]; then
9      echo "File has not found."
10     exit 1
11 fi
12
13 # if (amount-of-args) != 2
14 if [ $# -ne 2 ]; then
15     echo "There have to be 2 args."
16     exit 1
17 fi
18
19 echo "First number: $1"
20 echo "Second number: $2"
21
22 # construction $((smth)) evaluates expression
23 SUM=$(( $1 + $2 ))
24
25 echo "SUM: $SUM" | tee -a $FILE
```

```
Kovalyov.txt
1  Kovalyov Oleksandr Oleksiyovuch
2  Academic group = TP-12
3  Hobbies: Programming, learning Linux
4  SUM: 15
5
```

```
alex@debian:~/Documents/Lab5$ bash v29.sh
-- Var 29 --
There have to be 2 args.
alex@debian:~/Documents/Lab5$ bash v29.sh 5 10
-- Var 29 --
First number: 5
Second number: 10
SUM: 15
alex@debian:~/Documents/Lab5$
```

Варіант 30: Ввести з командного рядка вашу дату народження і записати її до вашого файлу.

Перевіряємо, чи існує файл. Зчитуємо з клавіатури день народження. Записуємо у файл за допомогою оператора перенаправлення з доповненням (<<).



```
$ v30.sh
1  #!/bin/bash
2
3  echo "-- Var 30 --"
4
5  FILE="Kovalyov.txt"
6
7  # is file exist?
8  if [[ !(-f $FILE) ]]; then
9      echo "File has not found."
10     exit 1
11 fi
12
13 # command "read" reading input from star
14 # -p key allows to print text input prom
15 read -p "Input your birthday, please: "
16
17 # >> - appending pipe
18 echo "Birthday: $BIRTHDAY" >> $FILE
19 echo "Done!"
```

```
1  Kovalyov Oleksandr Oleksiyovuch
2  Academic group = TP-12
3  Hobbies: Programming, learning Linux
4  SUM: 15
5  Birthday: 18-04-2003
6
```

```
alex@debian:~/Documents/Lab5$ bash v30.sh
-- Var 30 --
Input your birthday, please: 18-04-2003
Done!
alex@debian:~/Documents/Lab5$
```

Контрольні запитання:

1) Що таке скрипт?

Скрипт – це звичайний текстовий файл, що містить системні або вбудовані команди оболонки. У нашому випадку використовує інтерпретатор bash для виконання. Скрипти потрібні для виконання певних задач та їх автоматизації.

2) Як перетворити скрипт у виконуваний файл?

Для цього потрібно змінити режим файлу на виконуваний. Це можна зробити за допомогою команди `chmod`. Аргументом потрібно вказати тип користувачів, яким потрібно надати можливість запускати файл, і вказати ключ `x` (executable) та назву файлу.

3) Що означає символ «./», введений в командному рядку перед натисканням Enter?

Це означає те, що виконуваний каталог запускається з поточної директорії. Це робиться з урахуванням правил безпеки. Якщо б виконувані файли можна було запускати без цього, то це є вразливістю – можна залишити шкідливе програмне забезпечення під назвою однієї з частовживаних команд `bash`, наприклад, `ls`, і викликався би саме файл, а не потрібна команда.

4) Для чого використовується команда **read**?

Ця команда використовується для зчитування з клавіатури певних даних у змінну. Якщо вказати ключ **-s**, то буде використовуватись режим «введення паролю», і нічого не буде видно. З ключем **-r** можна додати запрошення для введення тексту. Після цього всього треба залишити назву змінної, в яку запишеться текст.

5) Як працює умовний оператор **if-fi**?

Умовний оператор **if-fi** перевіряє певну умову, після чого запускає гілку **then** або **else (elseif)**. Може використовувати при перевірці квадратні дужки **[]**, тобто вбудовану команду **test**. Також, якщо потрібні логічні оператори **АБО**, **І**, **НЕ** – ставляться подвійні квадратні скобки **[[]]** для уникнення помилок. Конструкція закривається ключовим словом **fi**.

6) Які конструкції застосовуються для організації циклу?

Для звичайних циклів найчастіше використовується конструкція **while-do-done**, трохи рідше – **until-do-done**. Окрім цього, частовживаною є конструкція **for-do-done**, яка дозволяє перебирати набір даних, наприклад, всі файли у директорії, тощо.

Висновок: за результатами виконання цієї лабораторної роботи було ознайомлено зі скриптовою мовою програмування **Bash**. Були освоєні базові конструкції мови, проведена робота з інтерпретатором, командами, операторами, конвейерами, функціями. Також, був повторений матеріал з використання утиліт **mc**, **apt**, **dpkg** і так далі.

Додаткові джерела:

- 1) [AskUbuntu - Do file extensions have any purpose in Linux?](#)
- 2) Advanced Bash-Scripting Guide – Mendel Cooper, 2004
- 3) [StackOverflow - How does the #! shebang work?](#)
- 4) [LinuxHint - Bash Variable Name Rules: Legal and Illegal](#)