

Семинар #7: Скрипты языка Bash. Практика.

Как сдавать задачи

Для сдачи ДЗ вам нужно создать репозиторий на GitLab (если он ещё не создан) под названием `devtools-homework`. Структура репозитория должна иметь вид:

```
├── seminar7_bash_scripting/
│   ├── 01.sh
│   ├── 02.sh
│   └── ...
└── ...
```

Для каждой задачи, если в самой задаче не сказано иное, нужно создать один скрипт с расширением `.sh` и шевбангом в начале скрипта. Если задача делится на подзадачи, нужно, если в самой задаче не сказано иное, создать скрипт для каждой подзадачи. Названия файлов решений для всех задач/подзадач должны начинаться с номера задачи, например `01.sh` или `04b.sh`, даже если в условии задачи используется другое имя для скрипта.

Файлы для этого задания можно найти в репозитории в папке `seminar7_bash_scripting/practice`.

Переменные

Задача 1. Новая переменная

Пусть есть следующая простая программа на языке Bash:

```
#!/bin/bash
echo "Hello, my name is Alice."
echo "Alice, welcome to the world of bash scripts!"
echo "Today Alice will learn about bash scripting."
```

Этот скрипт можно найти в `seminar7_bash_scripting/practice/hello_alice.sh`. Измените этот скрипт, добавив переменную `name` со значением `"Alice"` и три раза используйте эту переменную в командах `echo`.

Задача 2. Печатаем переменные среды

Напишите скрипт, который будет печатать следующие переменные среды, используя команду `echo`:

- `PATH` – переменная среды, которая содержит список каталогов, в которых оболочка ищет исполняемые файлы (команды).
- `USER` – переменная среды, которая содержит имя текущего пользователя.
- `UID` – переменная среды, которая содержит `UID` текущего пользователя.
- `HOME` – имя домашней директории текущего пользователя.
- `SHELL` – имя текущей используемой оболочки.
- `SHLVL` – уровень текущей оболочки.
- `HOSTNAME` – имя компьютера, на котором выполняется программа.
- `LANG` – язык и локаль для сообщений программ, например `en_US.UTF-8` или `ru_RU.UTF-8`.
- `PS1` – переменная, которая содержит строку перед вводом команды. Что будет, если присвоить этой переменной значение `"$ "`? Перезайдите в терминал, если хотите восстановить старое значение переменной.

Используйте команду `env`, чтобы напечатать все переменные среды и убедиться, что ваш скрипт печатает корректные значения.

Задача 3. Создаём свою переменную среды

Выполните следующие команды в терминале. Для каждой команды, определите, что будет печататься на экран.

- (a) Создайте новую переменную по имени **ALPACA** со значением **apple**.

```
$ ALPACA=apple
```

- (b) Распечатайте значение переменной **ALPACA**:

```
$ echo "This is $ALPACA"
```

- (c) Создайте новый файл **print_alpaca.sh**, который будет содержать следующий код:

```
#!/bin/bash
echo "This is $ALPACA"
```

После этого дайте этому файлу права на исполнение:

```
$ chmod +x print_alpaca.sh
```

- (d) Определите, что напечатает следующая последовательность команд:

```
$ unset ALPACA
$ ALPACA=apple
$ ./print_alpaca.sh
```

Команда **unset** удаляет переменную **ALPACA**. После этого создаётся новая переменная со значением **apple**. Это делается для чистоты эксперимента.

- (e) При исполнении скрипта в оболочке создаётся новый процесс и обычные переменные оболочки в этом процессе не видны. Но видны экспортированные переменные, создаваемые командой **export**. Определите, что напечатает следующая последовательность команд:

```
$ unset ALPACA
$ export ALPACA=apple
$ ./print_alpaca.sh
```

- (f) Можно задать экспортированные переменные для одной команды с помощью специального синтаксиса:

```
$ VAR1=value1 VAR2=value2 command
```

Определите, что напечатает следующая последовательность команд:

```
$ unset ALPACA
$ ALPACA=apple ./print_alpaca.sh
$ ALPACA=apple; ./print_alpaca.sh
$ ALPACA=apple && ./print_alpaca.sh
```

- (g) Команда **source** нужна для того, чтобы выполнить команды из скрипта в текущей оболочке, не создавая новый процесс. Определите, что будут печатать следующие команды:

```
$ unset ALPACA
$ ALPACA=apple
$ source ./print_alpaca.sh
```

- (h) Создайте экспортируемую переменную и убедитесь, что она работает:

```
$ export ALPACA=apple
$ ./print_alpaca.sh
```

После этого закройте терминал и заново его откройте. Что теперь напечатают команды:

```
$ ./print_alpaca.sh
$ source ./print_alpaca.sh
```

- (i) Добавьте создание переменной среды **ALPACA** в файл `~/.bashrc`. В этом файле нужно добавить строку:

```
export ALPACA=apple
```

Скрипт `.bashrc` выполняется в процессе текущей оболочки при каждом запуске терминала. После этого закройте терминал и заново его откройте. Что теперь напечатает команда:

```
$ ./print_alpaca.sh
```

- (j) Распечатайте все переменные среды, с помощью команды **env** и убедитесь, что добавилась новая переменная среды **ALPACA**.

Для того чтобы сдать эту задачу создайте файл `03.txt` в котором будут печататься выводы команд на каждом шаге в следующем формате:

```
# Subtask a
$ ALPACA=apple
# Subtask b
$ echo "This is $ALPACA"
This is apple
# Subtask c
...
```

Задача 4. Внутренняя или внешняя

Часть команд, которые используются в **bash** являются встроенными командами **bash** (например, команды **cd**, **echo** и другие), а некоторые команды являются отдельными программами (например, **ls**, **cp** и другие). Некоторые команды реализованы обоими способами, в этом случае при выполнении команды обычно выбирается встроенная версия. Чтобы понять какая команда является встроенной, а какая команда является отдельной программой, можно использовать команду **type -a**. Определите, как реализованы следующие команды:

- | | |
|----------------|-----------------|
| • cd | • mkdir |
| • cp | • echo |
| • pwd | • source |
| • ls | • type |
| • touch | • which |

Для решения этой задачи создайте файл `04.txt`, в котором будут представлены ответы в следующем формате:

```
cd - shell builtin
cp - program /usr/bin/cp
...
```

Если команда реализована несколькими способами, то выбирайте первый – наиболее приоритетный.

Задача 5. Создаём новую команду

- (a) Когда вы запускаете команду, если она не является встроенной, **bash** ищет эту программу в стандартных путях, задаваемых переменной **PATH**. Переменная **PATH** является строкой, в которой перечислены директории, разделённые символом `:`. Очистите переменную **PATH**, присвоив ей пустую строку.

```
$ export PATH=""
```

убедитесь, что после этого внешние команды перестали работать. **Перезапустите терминал, чтобы восстановить значение переменной PATH.**

- (b) Создайте простой скрипт `hello.sh`, который будет содержать следующий код:

```
#!/bin/bash
echo "Hello, Bash"
```

Дайте этому файлу права на исполнение и исполните его

```
$ chmod +x ./hello.sh
$ ./hello.sh
```

- (с) Попробуйте исполнить этот файл без указания текущей директории:

```
$ hello.sh
```

Без указания пути до файла `bash` будет искать такой файл в стандартных путях. Так как такого файла там не будет, команда не будет найдена.

- (d) Переместите данный скрипт в один из стандартных путей из переменной `PATH`, например, в `/usr/local/bin`. Для этой операции понадобятся права суперпользователя. Убедитесь, что у скрипта есть право на исполнение для текущего пользователя, и, желательно, для всех пользователей. После этого запустите скрипт без указания директории:

```
$ hello.sh
```

Теперь этот файл должен найтись, и скрипт запустится.

- (е) Измените имя скрипта в системной директории с `hello.sh` на `hello`. Теперь скрипт можно будет запустить, как любую другую команду.

```
$ hello
```

Для того чтобы сдать эту задачу создайте файл `05.sh` в котором нужно будет напечатать все команды, которые были исполнены при выполнении данного задания в следующем формате:

```
# Subtask a
$ export PATH=""
# Subtask b
...
```

Задача 6. Подмена `cat`

В этой задаче будет нужно подменить команду `cat` так, чтобы она вместо того, чтобы печатать содержимое файла/файлов, печатала на экран одно слово `"Meow"`, независимо от переданных ей аргументов. Добейтесь подобного поведения, изменив переменную `PATH`. После выполнения задания, перезагрузите терминал, чтобы вернуть прежнее значение переменной `PATH`.

Для того чтобы сдать эту задачу создайте файл `06.sh` в котором нужно будет напечатать все команды, которые были исполнены при выполнении данного задания.

Аргументы

Задача 7. Привет скрипт

Напишите программу на языке `bash`, которая будет печатать имя своего файла. Например, если скрипт вызывается как `./script.sh`, то он должен печатать на экран строку `"Hello ./script.sh"`.

```
$ ./script.sh
Hello ./script.sh
$ mv script.sh 07.sh
$ ./07.sh
Hello ./07.sh
```

Задача 8. Количество аргументов

Напишите программу, которая будет печатать количество переданных ей аргументов.

```
$ ./script.sh
0
$ ./script.sh apple
1
$ ./script.sh apple banana cherry durian
4
```

Задача 9. Первый аргумент

Напишите программу, которая будет печатать первый переданный ей аргумент.

```
$ ./script.sh

$ ./script.sh apple
apple
$ ./script.sh cherry durian
cherry
```

Задача 10. Все аргументы

Напишите программу, которая будет печатать все переданные ей аргументы через пробел. Используйте "\$@".

```
$ ./script.sh apple banana cherry durian
apple banana cherry durian
```

Кавычки

Задача 11. Одинарные или двойные кавычки

Определите, что сделают следующие команды в bash:

- | | |
|------------------------------|-----------------------------------|
| (a) touch alpha beta gamma | (j) echo I am in \$(pwd) folder |
| (b) touch "alpha beta gamma" | (k) echo "I am in \$(pwd) folder" |
| (c) touch 'alpha beta gamma' | (l) echo 'I am in \$(pwd) folder' |
| (d) echo /var/* | (m) echo I have \$100 |
| (e) echo "/var/*" | echo I have \\$100 |
| (f) echo '/var/*' | (n) echo "I have \$100" |
| (g) user=alice | echo "I have \\$100" |
| echo hello \$user | (o) echo 'I have \$100' |
| (h) user=alice | echo 'I have \\$100' |
| echo "hello \$user" | (p) touch hello.sh |
| (i) user=alice | find /usr -name *.sh |
| echo 'hello \$user' | (q) touch hello.sh |
| | find /usr -name "*.sh" |

Для того чтобы сдать эту задачу создайте файл 11.sh в котором нужно будет написать ответы на каждую из подзадач.

Коды возврата

Задача 12. Коды возврата базовых программ

В каких случаях следующие программы возвращают ненулевой код возврата:

- | | |
|-----------------------|----------------------------|
| (a) <code>ls</code> | (d) <code>grep</code> |
| (b) <code>diff</code> | (e) <code>id</code> |
| (c) <code>find</code> | (f) <code>ping -c 1</code> |

Вызовите каждую из этих команд 2 раза так, чтобы при первом вызове код возврата был равен нулю, а при втором вызове — не равен нулю. После каждого вызова напечатайте код возврата, используя команду

```
$ echo $?
```

Для того чтобы сдать эту задачу создайте файл `12.txt`, в котором напишите ответы на каждую из подзадач.

Задача 13. Коды возврата скобочных команд

Пусть заданы переменные:

```
$ a=apple  
$ x=80
```

В `bash` можно выполнить команду из двух квадратных скобок, содержащих условие. Такие команды ничего не выводят на экран, а только возвращают код возврата.

```
$ [[ $a == apple ]]  
$ echo $?  
0
```

Код возврата 0 говорит о том, что условие воспринимается как *истина*. Ненулевой код возврата говорил бы о том, что условие воспринимается как *ложь*.

Чему равны коды возврата следующих команд. Проверьте себя с помощью `echo $?`

- | | |
|---|--|
| (a) <code>[[\$a == apple]]</code> | (n) <code>((0))</code> |
| (b) <code>[[\$a != apple]]</code> | (o) <code>((1))</code> |
| (c) <code>[[\$a == banana]]</code> | (p) <code>((x + 1))</code> |
| (d) <code>[[\$a < banana]]</code> | (q) <code>((x - 80))</code> |
| (e) <code>[[! \$a < banana]]</code> | (r) <code>((x > 0))</code> |
| (f) <code>[[\$a > almond && \$a < cherry]]</code> | (s) <code>((x > 50 && x < 100))</code> |
| (g) <code>[[\$x == 80]]</code> | (t) <code>((x % 2))</code> |
| (h) <code>[[\$x -eq 80]]</code> | (u) <code>true</code> |
| (i) <code>[[\$x > 200]]</code> | (v) <code>false</code> |
| (j) <code>[[\$x -gt 200]]</code> | (w) <code>true && false</code> |
| (k) <code>[[0 == -0]]</code> | (x) <code>true false</code> |
| (l) <code>[[0 -eq -0]]</code> | (y) <code>false && echo Hello</code> |
| (m) <code>[[\$x < 50 \$x > 100]]</code> | (z) <code>true echo Hello</code> |

Для того чтобы сдать эту задачу создайте файл `13.txt`, в котором нужно будет указать код возврата для каждой из подзадач.

Ветвление

Важно! Используйте синтаксис с двойными скобками `if [...]`, а не с одинарными `if [...]`

Задача 14. Проверка количества аргументов

Напишите программу, которая будет проверять количество аргументов и печатать:

- **Too Little** – если в программу было передано меньше двух аргументов.
- **Correct** – если в программу было передано два или три аргумента.
- **Too Much** – если в программу было передано больше трёх аргументов.

Задача 15. Сравнение строк

Напишите программу, которая будет принимать две строки через аргументы и печатать:

- **Less** – если первая строка лексикографически меньше второй.
- **Equal** – если строки равны.
- **Greater** – если первая строка лексикографически больше второй.

Задача 16. Существует ли файл

Напишите скрипт, который принимает название файла через аргумент и печатает **Yes**, если такой файл (любого типа) существует в данной директории или **No**, если не существует. Решите эту задачу тремя способами:

- С использованием `if` и ключа `-e`.
- С использованием `if` и программы `ls`.
- С использованием операторов `&&`, `||` и программы `ls` (без использования `if`).

Скрипт должен принимать ровно один аргумент. Если количество передаваемых аргументов не равно одному, то программа должна завершаться с текстом:

```
Usage: $0 <filename>
```

Скрипт не должен выводить на экран никакого дополнительного текста, не указанного в условии задачи.

Задача 17. Удаление с сообщением

Напишите программу, которая принимает название файла через аргумент и пытается удалить этот файл. Если файл существует, то программа должна его удалить и напечатать **"File deleted"**. Если же файла не существует, то программа должна напечатать **"File does not exist"**. Решите задачу тремя способами:

- С использованием `if` и ключа `-f`.
- С использованием `if` и программы `ls`.
- С использованием операторов `&&` и `||` и программы `ls` (без использования `if`).

Задача 18. Есть ли право на исполнение?

Напишите программу, которая принимает путь к файлу и печатает:

- **Does not exist** – если файл не существует.
- **Yes** – если у файла есть право на исполнение для текущего пользователя.
- **No** – если у файла нет права на исполнение для текущего пользователя.

Задача 19. Проверка пользователя

Напишите программу, которая проверяет, существует ли пользователь. Программа должна принимать имя пользователя и возвращать код возврата 0, если пользователь существует и 1 в иных случаях. Программа не должна ничего печатать. Используйте программу `grep` и файл `/etc/passwd`. Ниже представлен пример использования программы в предположении, что в системе существуют пользователи `root` и `alice` и не существуют пользователи `tom`, `bash`.

```
$ ./script.sh root
$ echo $?
0
$ ./script.sh alice
$ echo $?
0
$ ./script.sh tom
$ echo $?
1
$ ./script.sh bash
$ echo $?
1
```

Циклы

Задача 20. Повторение

Напишите программу, которая будет печатать переданный ей аргумент 5 раз, используя цикл `for`.

```
$ ./script.sh apple
apple
apple
apple
apple
apple
```

Задача 21. Создаём файлы

Напишите программу, которая принимает число `N` и создаёт файлы с именами `data1.txt`, `data2.txt` ... `dataN.txt`. В каждом из файлов нужно написать соответствующее число. То есть в файл `data1.txt` нужно написать 1, в файл `data2.txt` нужно написать 2 и т. д. Используйте цикл `for`.

Задача 22. Аргументы с пробелами

Пусть есть следующая программа на языке `bash`:

```
#!/bin/bash
echo "1: "
for elem in $*; do echo $elem; done

echo
echo "2: "
for elem in $@; do echo $elem; done

echo
echo "3: "
for elem in "$*"; do echo $elem; done

echo
echo "4: "
for elem in "$@"; do echo $elem; done
```


Что напечатает эта программа, если её вызвать следующим образом:

```
$ ./script.sh apple banana 'cherry and durian'
```

Для решения этой задачи нужно создать файл в формате `.txt`, который будет содержать вывод программы.

Задача 23. Цикл по всем аргументам

Напишите программу, которая будет печатать все переданные ей аргументы, каждый аргумент в новой строке:

```
$ ./script.sh

$ ./script.sh apple
apple

$ ./script.sh apple banana cherry durian
apple
banana
cherry
durian

$ cd space_test
$ ../script.sh *
alpaca.txt
bison.txt
camel carries goods.txt
```

Директория `space_test` содержит файл, в имени которого содержатся пробелы.

Решите эту задачу тремя способами:

- (a) Используя цикл `for` и специальный массив из аргументов `"$@"`.
- (b) Используя цикл `while` и целочисленный счётчик.
- (c) Используя цикл `while` и команду `shift`.

Задача 24. Перенаправление цикла

- (a) Решите подзадачу 23(a), но вместо вывода на экран цикл `for` должен выводить результат в файл `resulta.txt`.
Решите эту подзадачу, используя только одно простое перенаправление `>`, применив его ко всему циклу.
- (b) Решите подзадачу 23(b), но вместо вывода на экран цикл `while` должен выводить результат в файл `resultb.txt`.
Решите эту подзадачу, используя только одно простое перенаправление `>`, применив его ко всему циклу.

Задача 25. Печать всех путей из PATH

Переменная `PATH` является строкой, которая содержит пути, разделённые символом `:`. Используйте цикл `for` и переменную `IFS`, чтобы напечатать все пути из этой переменной, каждый путь в новой строке.

```
$ ./script.sh
/usr/local/bin
/usr/bin
/bin
/usr/local/games
/usr/games
```

На вашей системе пути в `PATH` могут быть другими.

Группировка команд { ... } и сабшеллы (...)

Задача 26. Различие между группой команд и сабшеллом

(a) Что выведет на экран следующий скрипт?

```
#!/bin/bash
cd /home
a=apple
{
    cd /etc
    a=avocado
    pwd
    echo $a
}
pwd
echo $a
```

(b) Что выведет на экран следующий скрипт?

```
#!/bin/bash
cd /home
a=apple
(
    cd /etc
    a=avocado
    pwd
    echo $a
)
pwd
echo $a
```

Для решения этой задачи создайте текстовый файл, содержащий вывод скриптов.

Работа со строками

Для задач этой главы используйте возможности, описанные в файле теории `theory_bash_scripting.pdf` в главе "Манипуляция со строками".

Задача 27. Операции над строкой

Напишите программу, которая принимает одну строку через аргументы и печатает следующее:

- (a) Длину этой строки
- (b) Первые три символа строки
- (c) Последние три символа строки
- (d) Строка, в которой все подстроки "apple" заменены на "banana"
- (e) Строка, в которой все заглавные буквы заменены на соответствующие строчные буквы

Задача 28. Базовое имя

В Linux есть команда `basename`, которая принимает на вход полный путь до некоторого файла, а возвращает только имя этого файла.

```
$ basename /etc/passwd
passwd
$ basename /home/user/workspace/file.txt
file.txt
```

Вам нужно написать скрипт, который будет делать то же самое, но без использования программы `basename`.

Задача 29. Домен верхнего уровня

Напишите программу, которая будет принимать строку, являющуюся веб-адресом, через аргументы и печатать домен верхнего уровня этого веб-адреса.

```
$ ./script.sh www.google.com
com
$ ./script.sh https://tldp.org/LDP/abs/html/string-manipulation.html
org
$ ./script.sh ftp://sub.example.co.uk/path
uk
```

Управляющая конструкция case esac

Задача 30. Количество дней в месяце

Напишите программу, которая бы печатала количество дней в месяце по его короткому имени. Используйте конструкцию `case esac`.

```
$ ./script.sh jan
31
$ ./script.sh feb
28 or 29
$ ./script.sh apr
30
```

Задача 31. Печать текста с опциями

Напишите программу, которая бы делала простейшие манипуляции со строкой в зависимости от переданных ей опций. Программа должна принимать опцию и строку через аргументы командной строки. Программа должна поддерживать следующие операции над строкой:

<code>--lower</code>	перевод всех букв в нижний регистр
<code>--upper</code>	перевод всех букв в верхний регистр
<code>--length</code>	вычисление длины строки
<code>--reverse</code>	обращение строки
<code>--prefix n</code>	возвращение только первых <code>n</code> символов строки

Примеры работы такой программы:

```
$ ./script.sh --lower APPLE123
apple123

$ ./script.sh --upper banana123
BANANA123

$ ./script.sh --length cherry
6

$ ./script.sh --reverse durian
nairud

$ ./script.sh --prefix 3 eggplant
egg
```

Работа с целыми числами

Задача 32. Куб

Напишите программу, которая принимает число через аргументы и печатает куб этого числа.

```
$ ./script.sh 5
125
```

Задача 33. Квадраты

Напишите программу, которая будет печатать квадраты первых 100 натуральных чисел. Используйте цикл `for` вместе с `((...))`.

Задача 34. Сумма цифр

Напишите программу, которая принимает целое число и возвращает сумму цифр этого числа.

Массивы

Задача 35. Работа с массивом

Пусть есть следующий массив:

```
array=(apple banana cherry durian eggplant fig guava)
```

Произведите над массивом следующие операции:

- (a) Напечатайте первый элемент массива
- (b) Напечатайте последний элемент массива
- (c) Напечатайте количество элементов в массиве
- (d) Напечатайте все элементы массива
- (e) Удалите элемент с индексом 1 (`banana`)
- (f) Добавьте строку `"hazelnut"` в конец массива
- (g) Добавьте строку `"fruit"` после элемента с индексом 4

Словари

Задача 36. Работа со словарём

Пусть есть следующий словарь:

```
declare -A map
map["alpaca"]="apple"
map["bison"]="banana"
map["camel"]="cherry"
map["dog"]="durian"
map["eagle"]="eggplant"
map["fox"]="fig"
```

Произведите над словарём следующие операции:

- (a) Напечатайте значение с ключом `"dog"`
- (b) Напечатайте количество пар ключ-значение в словаре
- (c) Напечатайте все значения словаря

- (d) Напечатайте все ключи словаря
- (e) Используйте цикл `for`, чтобы напечатать все пары ключ-значение в формате:
- ```
Key: alpaca, Value: apple
Key: bison, Value: banana
...
```
- (f) Добавьте в словарь пару (ключ="giraffe", значение="guava")
- (g) Удалите из словаря пару с ключом "dog".

## Функции

### Задача 37. Факториал

Напишите программу, содержащую функцию `fact`. Эта функция должна принимать число через аргументы и возвращать факториал этого числа через `echo`. Если на вход приходит отрицательное число, то код возврата функции должен быть 1, в ином случае – 0.

### Задача 38. Тип файла

Напишите программу, содержащую функцию `get_file_type`. Эта функция должна принимать путь до файла и возвращать через `echo` строку, описывающую тип этого файла, а именно она должна возвращать:

|                     |                                         |
|---------------------|-----------------------------------------|
| "regular"           | если это обычный файл                   |
| "directory"         | если этот файл – директория             |
| "symbolic link"     | если это символическая ссылка           |
| "block special"     | если это файл блочного устройства       |
| "character special" | если это файл символьного устройства    |
| "other"             | если тип файла ни один из перечисленных |

Функция должна иметь код возврата 1, если файл не существует, и 0 в ином случае.

### Задача 39. Массовое переименование

Напишите функцию `add_suffix`, которая будет массово переименовывать файлы, добавляя суффикс для имени каждого файла. Функция должна принимать суффикс через первый аргумент и пути до файлов через аргументы, начиная со второго. Например, вызов функции:

```
$ add_suffix ".backup" a.txt b.txt c.txt
```

должен переименовывать файлы `a.txt`, `b.txt`, `c.txt` в `a.txt.backup`, `b.txt.backup`, `c.txt.backup`. Функция должна переименовывать как обычные файлы, ссылки, так и директории.

Если какой-либо из переданных файлов не существует, то функция для этого файла должна вывести в `stderr`:

```
add_suffix: cannot access <filename>: No such file or directory
```

При этом все остальные файлы должны быть обработаны. Функция должна возвращать код возврата 0, если все файлы существуют и они были переименованы, и 1 в иных случаях.

## Чтение

### Задача 40. Сумма

Напишите программу, которая считывает два числа из стандартного входа, а потом печатает сумму этих чисел.

```
$./script.sh
Вводим первое число 10
Вводим второе число 20
30
```

## Задача 41. Чтение в тихом режиме

Напишите программу, которая будет читать строку из `stdin` в тихом режиме (`read -s`), а затем выводить эту строку на экран.

## Задача 42. Нумерация строк

Напишите программу, которая будет принимать через аргументы название файла, а затем печатать содержимое файла на экран, добавляя номер строки и количество символов в строке перед началом каждой строки. Например, если файл содержит:

```
Alpaca loves apples.
Bison can run up to 35 miles per hour.
Camel is a very unique mammal.
Dog barks very loudly.
```

то программа должна вывести:

```
[1: 20] Alpaca loves apples.
[2: 38] Bison can run up to 35 miles per hour.
[3: 30] Camel is a very unique mammal.
[4: 22] Dog barks very loudly.
```

Решите эту задачу, используя цикл `while` и команду `read`.