

# Лабораторная работа № 2.

## Командная оболочка bash:

### перенаправление ввода-вывода, сценарии и другие

### ВОЗМОЖНОСТИ

#### Table of Contents

1. Методические материалы.....	3
1.1. Перенаправление ввода вывода.....	3
1.2. Использование утилиты awk.....	4
1.3. Поточковый редактор sed.....	7
1.4. Командная оболочка bash.....	8
Что хранится в окружении?.....	8
Что такое сценарий командной оболочки?.....	9
Как написать сценарий командной оболочки?.....	10
Формат файла сценария.....	10
Разрешения на выполнение.....	11
Местоположение файла сценария.....	11
2. Задания лабораторной работы.....	14
Критерии оценивания.....	14
Содержание отчета.....	14
Задание 2.1. Перенаправление ввода-вывода.....	14
Примеры к заданию 2.1.....	15
Задание 2.2. Утилита awk примеры для изучения.....	17
Примеры к заданию 2.2.....	17
Пример 1.....	17
Пример 2.....	18
Пример 3.....	18

Пример 4.....	19
Пример 5.....	19
Пример 6.....	20
Пример 7.....	20
Пример 8.....	20
Пример 9.....	20
Пример 10.....	20
Пример 11.....	20
Пример 12.....	20
Пример 13.....	20
Пример 14.....	21
Пример 15.....	21
Задание 2.3. Обработка строк файла с применением потокового редактора sed.....	21
Примеры к заданию 2.3.....	22
Задание 2.4. Исполнительная среда оболочки Bash.....	23
2.4.1. Примеры для изучения: Исполнительная среда оболочки.....	23
2.4.2. Примеры для изучения: Исполнение команд.....	25
2.4.3. Примеры для изучения: Сценарии Bash.....	25
Пример 1.....	26
Пример 2.....	26
Пример 3.....	26
Пример 4.....	27
Пример 5.....	27
Пример 6.....	27
Пример 7.....	28
Пример 8.....	28
Пример 9.....	29
Пример 10.....	29
Пример 11.....	29
Пример 12.....	29
Задание 2.5. Для самостоятельной работы.....	29
Варианты.....	30

Вариант 0.....	30
Вариант 1.....	31
Вариант 2.....	32
Вариант 3.....	32
Вариант 4.....	33
Вариант 5.....	34
Вариант 6.....	35

## 1. Методические материалы

### Цели работы:

- Изучить синтаксис команд перенаправления ввода-вывода.
- Изучить утилиту `awk`, ее функциональные возможности и синтаксис.
- Изучить потоковый редактор `sed` и его возможности.
- Изучить язык сценариев `bash`

### 1.1. Перенаправление ввода вывода

Многие утилиты в среде UNIX работают с потоками ввода и вывода. При этом у каждой программы есть стандартный поток ввода (обычно клавиатура) стандартный поток вывода (монитор) и стандартный поток вывода ошибок (тоже монитор). Во многих оболочках имеется возможность изменять стандартные потоки, например, чтобы программа читала данные не с клавиатуры, а из файла, или чтобы программа выводила данные не на экран, а сразу записывала их в файл. Кроме того, можно сделать так, чтобы поток вывода одной программы являлся потоком ввода другой. Эти механизмы называются перенаправлением потоков и значительно увеличивают возможности системы по манипулированию различными данными.

Рассмотрим этот механизм на конкретных примерах.

```
$cat file.my > file.my.new
```

команда `cat file.my` должна вывести содержимое файла `file.my` на экран, но в данном случае мы указали, чтобы все данные были помещены в файл `file.my.new` с помощью спецсимвола `>` - перенаправления стандартного потока вывода. Таким образом, мы получили еще один способ копирования файлов.

```
$grep name file.my > names
```

— сохраняет все строки файла `file.my`, содержащие сочетание букв `"name"` в файле `names`.

При этом предыдущее содержимое файла `names` будет удалено, чтобы

избежать этого можно добавить новые данные в конец файла с помощью символов ">>"

`$grep name file.my >> names` — новые данные будут добавлены в конец файла `names`.

Для перенаправления потока ввода следует использовать символ "<"

`$sort < file.my` — отсортировать строки в файле `file.my` и вывести на экран.

Перенаправление можно комбинировать

`$sort <file.my > file.my.sorted` — отсортировать и сохранить в файле `file.my.sorted`

Для перенаправления потока вывода ошибок можно использовать символы "2>"

`$prog 2>errors` — запустить программу `prog` и все ошибки сохранить в файле `errors`

Потоки вывода и ошибок можно совместить

`$ prog 2>&1 > output` — перенаправлять поток ошибок (2) туда же, куда и поток вывода (1), т.е. в файл `output`.

С помощью символа "|" можно организовать перенаправление потока вывода одной программы в поток ввода другой.

`$cat file.my | grep name > names` — содержимое файла `file.my` направить команде `grep name` и результат работы последней сохранить в файле `names`.

Такая составная команда называется **конвейер**. В конвейер можно объединять произвольное число команд.

`$cat file.my | grep name | wc -l` — подсчитать количество строк в файле `file.my`, в которых встречается последовательность символов "name".

`$cat words | uniq | sort` — вывести в отсортированном виде содержимое файла `words` без дубликатов (команда `uniq` удаляет дубликаты).

Стоит отметить, что отдельной программы выполняющей такую функцию нет, но благодаря использованию конвейера такой результат можно получить одной командой.

## 1.2. Использование утилиты **awk**

Утилита **awk** предназначена для нахождения информации в текстовых файлах по заданным критериям выбора. Она снабжена мощными средствами обработки текста в больших текстовых файлах. Утилиту **awk** можно отнести к программируемым фильтрам, настроенным на выполнение конкретной задачи.

**awk** одновременно является утилитой, программируемым фильтром и средой программирования, с помощью которой можно создавать другие фильтры. Поэтому она занимает особое место в операционных системах ОС Unix и ОС Linux и является мощным и гибким инструментом. **awk** является версией утилиты **awk**, используемой в UNIX и разработанной одним из создателей языка C Брайеном Керниганом и другими.

Утилиту **awk** можно использовать для создания отчетов, поиска заданных текстовых фрагментов, выполнения вычислений на основе вводимых данных, для работы со структурами, напоминающими базы данных, т.е. состоящих из записей, подразделяющихся на поля, разделенные специальными символами.

Схема вызова **awk** выглядит так:

```
$ awk options program file
```

**awk** воспринимает поступающие к нему данные в виде набора записей. Записи представляют собой наборы полей. Упрощенно, если не учитывать возможности настройки **awk** и говорить о некоем вполне обычном тексте, строки которого разделены символами перевода строки, запись — это строка. Поле — это слово в строке.

Рассмотрим наиболее часто используемые ключи командной строки **awk**:

- **-F fs** — позволяет указать символ-разделитель для полей в записи.
- **-f file** — указывает имя файла, из которого нужно прочесть **awk**-скрипт.
- **-v var=value** — позволяет объявить переменную и задать её значение по умолчанию, которое будет использовать **awk**.
- **-mf N** — задаёт максимальное число полей для обработки в файле данных.
- **-mr N** — задаёт максимальный размер записи в файле данных.
- **-W keyword** — позволяет задать режим совместимости или уровень выдачи предупреждений **awk**.

Утилиту **awk** можно вызвать непосредственно из командной строки или из shell-сценария с помощью ключевого слова **awk**. Если сценарий включает **awk** как составляющую, то его можно рассматривать как новый фильтр, утилиту или команду.

Синтаксис команды **awk** следующий:

```
$ 'Шаблон или условие {действие}' имена_файлов
```

Командой **awk** используется шаблон поиска такой, какой применяется в командах-фильтрах **grep**, **fgrep**, **egrep**. Шаблон выделяется символами косой черты (знаками слеш).

*Пример 1*

```
$ awk 'интерфейс Gnome/ {print}' /home/info/*
```

Ищутся все файлы в каталоге **/home/info**, в которых встречается последовательность из слов **интерфейс Gnome** и результаты направляются на стандартное устройство вывода — действие **{print}**. В результатах выводятся имена файлов и строки, содержащие шаблон.

*Пример 2. Найти по указанному пути файлы исходных текстов на языке C, включающие слова, обозначающие имя переменной, например, SHELLDATA или SHDATA:*

**\$ awk '/SHELLDATA|SHDATA/' /home/include/\***

*Пример 3. Предположим, что у нас есть текстовый файл /home/tab1/list\_students, отсортированный по алфавиту, каждая строка которого включает: фамилию, имя, факультет, курс, рейтинговая оценка, и состоящий из следующих строк-записей:*

Аринин Иван МП 1 4  
Бетинов Евгений ЭКТ 2 5  
Кошин Леонид МП 1 4  
Кошкин Владимир ЭКТ 1 5  
Липин Федор МП 2 3  
Пенов Николай МП 1 4  
Яшин Петр ЭКТ 2 4

*Требуется вывести список, состоящий из фамилий, имен и рейтинговых оценок.*

Нужно вывести 1, 2, 5 поля. Поля перечисляются через запятую и пробел, команда выглядит следующим образом:

**\$ awk '{print \$1, \$2, \$5}' /home**

Скорее всего, Вы находитесь в каталоге **/home**, поэтому имя файла достаточно описать так: **/tab1/list\_students**. В результате выполнения на экран выводится следующая таблица, поля которой будут выровнены.

Аринин Иван 4  
Бетинов Евгений 5  
Кошин Леонид 4  
Кошкин Владимир 5  
Липин Федор 3  
Пенов Николай 4  
Яшин Петр 4

*Пример 4. Теперь распечатаем весь файл в виде таблицы, т.е. чтобы все поля были выровнены. Для этого используем переменную **\$0**, иначе выведенные данные не выравниваются и будут плохо читаемы:*

**\$ awk '{print \$0}' /home/tab1/list\_students**

Результат вывода на экран будет следующий:

Аринин Иван МП 1 4

Бетинов Евгений ЭКТ 2 5  
Кошин Леонид МП 1 4  
Кошкин Владимир ЭКТ 1 5  
Липин Федор МП 2 3  
Пенов Николай МП 1 4  
Яшин Петр ЭКТ 2 4

*Пример 5.* Выберем из исходного файла `/home/tab1/list_students` студентов с рейтинговой оценкой 5 и распечатаем все значения полей для выбранных записей:

```
$ awk '/5/ {print $0}' /home/tab1/list_students
```

Бетинов Евгений ЭКТ 2 5  
Кошкин Владимир ЭКТ 1 5

В случае, если 5 встречается в других полях, например, курс 5, то условия выборки недостаточны.

*Пример 6.* Выбрать из списка студентов факультета ЭКТ, 1 курса:

```
$ awk '($3 == "ЭКТ") && ($4 == 1) {print}' list_student
```

Подробнее применение утилиты `awk` рассматривается в материале, опубликованном в курсе «Программирование мобильных и встраиваемых систем» по ссылке <https://edufpmi.bsu.by/mod/resource/view.php?id=736>.

### 1.3. Поточковый редактор `sed`

Команда `sed` (сокращение от `stream editor` — «редактор потока») является автоматическим текстовым редактором, который принимает входящий поток (файл или стандартный ввод), изменяет его в соответствии с некоторым выражением и выводит результат в стандартный вывод. Во многих отношениях команда `sed` подобна команде `ed`, первичному текстовому редактору Unix. Она обладает множеством операций, инструментами подстановки и возможностями работы с адресацией.

Хотя команда `sed` является довольно большой и ее детальное рассмотрение выходит за рамки курса, легко понять, как она устроена. В общих чертах, команда `sed` воспринимает адрес и операцию как один аргумент. Адрес является набором строк, и команда решает, что делать с этими строками.

Очень распространенная задача для команды `sed` : заменить какое-либо регулярное выражение текстом, например, так:

```
$ sed 's/exp/text/'
```

Так, если требуется заменить первое двоеточие в файле `/etc/passwd` на символ `%`, а затем отправить результат в стандартный вывод, следует

выполнить следующую команду:

```
$ sed 's/:/%/' /etc/passwd
```

Чтобы заменить все двоеточия в файле `/etc/passwd`, добавьте спецификатор `g` в конце операции, как в примере ниже:

```
$ sed 's/:/%/g' /etc/passwd
```

Рассмотрим команду, которая работает построчно; она считывает файл `/etc/passwd` и удаляет строки с третьей по шестую, а затем отправляет результат в стандартный вывод:

```
$ sed 3,6d /etc/passwd
```

В этом примере число `3,6` является адресом (диапазоном строк), а флаг `d` — операцией (удаление). Если адрес опустить, команда `sed` будет работать со всеми строками входного потока. Двумя самыми распространенными операциями команды `sed` являются `s` (найти и заменить) и `d`.

В качестве адреса можно также использовать регулярное выражение. Эта команда удаляет любую строку, которая соответствует регулярному выражению `exp` :

```
$ sed '/exp/d'
```

Подробнее применение потокового редактора `sed` рассматривается в материале, опубликованном в курсе «Программирование мобильных и встраиваемых систем» по ссылке <https://edufpmi.bsu.by/mod/resource/view.php?id=463>.

## 1.4. Командная оболочка `bash`

**Оболочка** — это интерпретатор команд. Это больше, чем просто изолирующий слой между ядром операционной системы и пользователем, это довольно мощный язык программирования.

### Что хранится в окружении?

Командная оболочка хранит в окружении данные двух основных типов, хотя `bash` практически не делает различий между типами. Эти данные хранятся в переменных окружения и в переменных командной оболочки. **Переменные командной оболочки** — это фрагменты данных, инициализируемые командой `bash`, а **переменные окружения** — практически все остальное. Помимо переменных командная оболочка хранит также программируемые данные, а именно псевдонимы и функции командной оболочки.

Можем использовать окружение для настройки некоторых параметров командной оболочки. Для этого используются следующие команды:

- `printenv` — выводит часть или все окружение;



- `set` — устанавливает параметры командной оболочки;
- `export` — экспортирует окружение для программ, которые будут выполняться;
- `alias` — создает псевдоним команды.

В таблице представлены примеры переменных окружения:

Переменная	Содержит
DISPLAY	Имя вашего дисплея, если вы работаете в графическом окружении. Обычно это <code>:0</code> , что означает первый дисплей, сгенерированный X сервером.
EDITOR	Имя программы, используемой в качестве текстового редактора.
SHELL	Имя командной оболочки.
HOME	Путь к домашнему каталогу.
LANG	Определяет набор символов и порядок сортировки для вашего языка.
PATH	Список каталогов, разделенных двоеточием, в которых производится поиск выполняемых программ по их именам.
PS1	Строка приглашения к вводу No 1. Определяет содержимое строки приглашения к вводу в командной оболочке.
PWD	Текущий каталог.
TERM	Тип терминала. Unix-подобные системы поддерживают множество протоколов для работы с терминалами; эта переменная определяет протокол, который будет использоваться при обмене данными с эмулятором терминала.
USER	Имя пользователя.

## Что такое сценарий командной оболочки?

Что такое сценарий командной оболочки? Выражаясь простым языком, **сценарий командной оболочки** — это файл, содержащий последовательность команд. Командная оболочка читает этот файл и выполняет команды, как если бы они вводились вручную в командной строке. **Командная оболочка** — это одновременно и мощный **интерфейс командной строки** системы, и **интерпретатор языка сценариев**. Как вы увидите далее, многое из того, что можно сделать в командной строке, также можно сделать в сценариях, а многое из того, что можно сделать в сценариях, можно сделать в командной строке. Практически весь набор команд, утилит и инструментов Linux доступен для вызова с помощью сценария оболочки.

Если этого не достаточно, то внутренние команды оболочки, такие как конструкции условий и циклов, придают сценариям дополнительную мощь и гибкость. Сценарии оболочки особенно хорошо подходят для решения задач управления системой и других рутинных, повторяющихся, задач.

## Как написать сценарий командной оболочки?

Чтобы успешно создать и запустить сценарий командной оболочки, нам нужно:

1. **Написать сценарий.** Сценарии командной оболочки — это обычные текстовые файлы. Поэтому для их создания нам понадобится текстовый редактор. Лучше использовать текстовый редактор, обладающий функцией подсветки синтаксиса, позволяющей видеть элементы сценариев с цветной маркировкой. Подсветка синтаксиса помогает замечать некоторые типичные ошибки. Для создания сценариев хорошо подходят vim, gedit, kate и многие другие редакторы.
2. **Сделать сценарий выполняемым.** Система не позволяет интерпретировать любой старый текстовый файл как программу, и небезосновательно! Поэтому, чтобы выполнить сценарий, файлу сценария нужно дать разрешения на выполнение.
3. **Поместить сценарий в каталог, где командная оболочка сможет найти его.** Командная оболочка автоматически выполняет поиск выполняемых файлов в нескольких каталогах, если путь к файлу не указан явно. Для максимального удобства необходимо помещать ваши сценарии в такие каталоги.

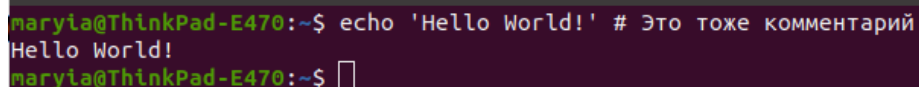
## Формат файла сценария

Следуя традициям программирования, напомним программу «hello world», чтобы продемонстрировать чрезвычайно простой сценарий. Итак, запустите текстовый редактор и введите следующий сценарий:

```
#!/bin/bash
# Это наш первый сценарий
echo 'Hello World!'
```

Последняя строка в сценарии хорошо знакома — это простая команда echo со строковым аргументом. Вторая строка — комментарий. Комментарии можем использовать и в командной строке:

```
$ echo 'Hello World!' # Это тоже комментарий
Hello World!
```



```
maryia@ThinkPad-E470:~$ echo 'Hello World!' # Это тоже комментарий
Hello World!
maryia@ThinkPad-E470:~$
```

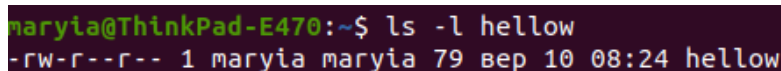
Первая строка в сценарии смотрится несколько необычно. Она похожа на комментарий, потому что начинается с символа #, но выглядит какой-то уж слишком специальной, чтобы быть комментарием. Последовательность символов `#!` — это на самом деле специальная конструкция, которая называется *shebang* (произносится как «ше-банг») и сообщает системе имя интерпретатора, который должен использоваться для выполнения следующего за ним текста сценария. Каждый сценарий командной оболочки должен включать это определение в первой строке.

Сохраните файл сценария из примера выше с именем *hellow*.

## Разрешения на выполнение

Далее сделаем сценарий исполняемым при помощи команды `chmod`. Просмотрим текущие права доступа

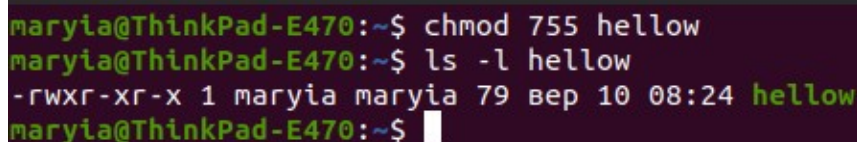
```
$ ls -l hellow  
-rw-r--r-- 1 maryia maryia 79 вер 10 08:24 hellow
```



Существует два распространенных набора разрешений для сценариев: **755** — для сценариев, которые должны быть доступны для выполнения всем, и **700** — для сценариев, которые могут выполняться только владельцами.

Добавим права на выполнение, используя команду `chmod`:

```
$ chmod 755 hellow  
$ ls -l hellow  
-rwxr-xr-x 1 maryia maryia 79 вер 10 08:24 hellow
```

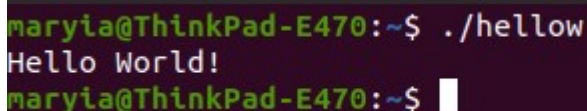


Обратите внимание, что сценарии необходимо сделать доступными для чтения, чтобы их можно было выполнить.

## Местоположение файла сценария

После установки разрешений попробуем запустить сценарий:

```
$ ./hellow
```



Но чтобы это сделать, необходимо добавить явный путь перед его именем. В противном случае мы получим следующее сообщение:

```
$ hellow
```

```
bash: hellow: команда не найдена
```

В чем причина? Чем наш сценарий отличается от других программ? Как оказывается, ничем. У нас замечательный сценарий. Его проблема — местоположение.

Напомним, что система просматривает каталоги по списку всякий раз, когда требуется найти исполняемую программу, если путь к ней не указан явно. Именно так система выполняет программу `/bin/ls`, если мы вводим `ls` в командной строке. Каталог `/bin` — один из каталогов, которые система просматривает автоматически. Список каталогов хранится в переменной окружения `PATH`. Она содержит список каталогов, перечисленных через двоеточие. Увидеть, что содержится в `PATH`, можно с помощью команды:

```
$ echo $PATH
```

```
/home/maryia/bin:/home/maryia/.local/share/umake/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/usr/lib/jvm/java-12-oracle/bin:/usr/lib/jvm/java-12-oracle/db/bin
```

Получим список каталогов. Если поместить сценарий в любой из этих каталогов, наша проблема будет решена. Обратите внимание на первый каталог в списке, `/home/maryia/bin`. В большинстве дистрибутивов Linux в переменную `$PATH` включается каталог `bin` в домашнем каталоге пользователя, чтобы дать пользователям возможность выполнять собственные программы.

Если каталога `bin` в домашнем каталоге в списке путей окружения нет, то достаточно создать его, добавить в список путей, поместить сценарий в него и можно будет запускать сценарий как любые другие программы:

```
$ mkdir bin
```

```
$ mv hellow bin
```

```
$ hellow
```

```
Hello World!
```

Если каталог отсутствует в переменной `PATH`, его легко туда добавить, включив следующую строку в файл `.bashrc`:

```
$ export PATH="$HOME/bin:$PATH"
```

Команда `export` экспортирует измененную переменную в дочерие среды процессов оболочки. Теперь вы можете запускать ваши скрипты,

просто набрав имя исполняемого скрипта без указания полного пути к исполняемому файлу.

Однако это изменение носит временный характер и действует только в текущем сеансе оболочки.

Чтобы сделать изменение постоянным, вам нужно определить переменную `$PATH` в файлах конфигурации оболочки.

В большинстве дистрибутивов Linux при запуске нового сеанса переменные среды считываются из следующих файлов:

- Конфигурационные файлы глобальной оболочки, такие как `/etc/profile`. Используйте этот файл, если вы хотите, чтобы новый каталог был добавлен всем системным пользователям `$PATH`.
- Конфигурационные файлы для отдельных пользовательских оболочек. Например, если вы используете Bash, вы можете установить переменную `$PATH` в файле `~/.bashrc`, а если вы используете Zsh – имя файла `~/.zshrc`.

В этом примере мы установим переменную в файле `~/.bashrc`. Откройте файл в текстовом редакторе и добавьте в конце следующую строку:

```
export PATH="$HOME/bin:$PATH"
```

Чтобы применить изменения в текущем сеансе, нужно заставить командную оболочку повторно прочитать файл `bashrc`, например, так:

```
$ . .bashrc
```

Или так:

```
$ source ~/.bashrc
```

Команда «точка» ( `.` ) в первом примере является синонимом `source`, встроенной команды, которая читает указанный файл и интерпретирует его как ввод с клавиатуры.

Дополнительные материалы по командной оболочке `bash` и разработке сценариев опубликованы в [тема 2](#) курса «Программирование мобильных и встраиваемых систем».

## 2. Задания лабораторной работы

### Критерии оценивания

#### Оценка 4

Выполнены 80% примеров заданий 2.1-2.4. Представлены файлы протоколов команд и меток времени с недочетами, исходный код скриптов. Лабораторная работа сдана с задержкой в 2 недели.

### Оценка 5-6

Выполнены 100% примеров заданий 2.1-2.4 и задачи 1-2 задания 2.5. Представлен отчет, файлы протоколов и меток времени, код скриптов в git-репозитории. Отчет, файлы протоколов команд и меток времени без ошибок. Лабораторная работа сдана с задержкой в 1 неделю.

### Оценка 7-8

Выполнены 100% примеров заданий 2.1-2.4 и задачи 1-4 задания 2.5. Представлен отчет, ответы на контрольные вопросы, файлы протоколов и меток времени, исходные коды скриптов в git-репозитории. Отчет, файлы протоколов команд и меток времени могут содержать незначительные ошибки. Лабораторная работа сдана с задержкой в 1 неделю.

### Оценка 9

Выполнены 100% примеров заданий 2.1-2.4 и задачи 1-5 задания 2.5 на отличном уровне. Представлен отчет, ответы на контрольные вопросы и файлы протоколов и меток времени, исходные коды скриптов в git-репозитории. Отчет, файлы протоколов команд и меток времени не содержат ошибок. Лабораторная работа сдана в срок.

## Содержание отчета

1. Цель работы.
2. Вариант задания.
3. Протоколы выполненных действий.
4. Исходные код скриптов

Отчет должен быть опубликован в git-репозитории.

## Задание 2.1. Перенаправление ввода-вывода

Изучите примеры задания 2.1 и выполните их в ОС Ubuntu и Raspberry PI (ранее Raspbian).

Включите ведение протокола командой `script` с журналом меток времени. **Протокол** назвать по следующему шаблону — `taskXФамилияNM`, где `X` — номер выполняемого задания, `Фамилия` — заменить на вашу фамилию латиницей и строчными буквами, `N` — номер группы, например 12 или 13, `M` — `r` — для Raspberry PI, `u` — для Ubuntu. **Журнал меток** назвать по следующему шаблону — `timelogXФамилияNM`, где `X` — номер выполняемого задания, `Фамилия` — заменить на вашу фамилию латиницей и строчными буквами, `N` — номер группы, например 12 или 13, `M` — `r` — для Raspberry PI, `u` — для Ubuntu.

## Примеры к заданию 2.1

### Выполнить примеры ниже в Raspberry PI

1. Создайте в текстовом редакторе два файла для дальнейшего использования в лабораторной работе. Файл `packages1.txt` должен содержать следующих восемь строк:

```
amanda  
galeon  
metacity  
mozilla  
postgresql  
procinfo  
rpmfind  
squid
```

Файл `packages2.txt` должен содержать следующих шесть строк:

```
anaconda  
openssh  
gnome-core  
samba  
sendmail  
xscreensaver
```

2. Для просмотра содержимого файлов предназначена команда `cat`. В простейшем случае эта команда принимает ввод из файла или стандартного канала ввода STDIN и посылает его в стандартный канал вывода STDOUT.

```
$ cat packages1.txt
```

3. Если команда `cat` не имеет аргументов, то ожидается, что она получает данные из стандартного канала ввода STDIN. Команда читает данные из канала стандартного ввода до нажатия на клавиатуре клавиш `<Ctrl+d>`:

```
$ cat
```

Type some sample text, then press return.

```
<Ctrl+d>
```

4. Большинство команд Linux работают как фильтры, т. е. Получают исходные данные из STDIN, делают обработку этих данных и выводят результаты в STDOUT. Например, команда `tr` (translate):

```
$ tr 'aeiou' 'AEIOU'
```

Type some sample text, then press return.

<Ctrl+d>

5. Используя директиву '>', перенаправим стандартный вывод из одного файла в другой файл:

```
$ cat packages1.txt > packages1.catfile
```

```
$ cat packages1.catfile
```

Затем сверим исходный и полученный файлы:

```
$ diff packages1.txt packages1.catfile
```

```
$ ls -l packages1*
```

6. Для присоединения существующего файла к другому файлу предназначена директива '>>':

```
$ cat packages2.txt >> packages1.catfile
```

```
$ cat packages1.catfile
```

**Подключиться по ssh из Raspberry PI и выполнить примеры ниже в Ubuntu**

7. Если команде cat не передается аргумент и стандартный ввод перенаправлен в файл, то весь ввод с клавиатуры до нажатия клавиш <Ctrl+d> будет перенаправлен в файл.

```
$ cat > typedin.txt
```

This time, when text is typed at the keyboard,  
it is redirected to the file typedin.txt.

<Ctrl+d>

```
$ ls -l typedin.txt
```

```
$ cat typedin.txt
```

8. Повторите предыдущий шаг, подставив вместо команды cat команду tr.

```
$ tr 'aeiou' 'AEIOU' > trfile.txt
```

This time, when text is typed at the keyboard,  
it is not echoed back to the screen with the translations made.

Instead, it is redirected to the file trfile.txt.

<Ctrl+d>

```
$ ls -l trfile.txt
```

```
$ cat trfile.txt
```

9. Команда cat принимает в качестве аргумента имя файла или стандартный



ввод, перенаправленный из файла. Проверьте это при помощи следующих двух команд:

```
$ cat packages1.txt
```

```
$ cat < packages1.txt
```

10. Однако команда `tr` принимает ввод только из стандартного канала.

```
$ tr 'aeiou' 'AEIOU' < packages1.txt
```

11. В следующем примере стандартный ввод и вывод одновременно перенаправляются

```
$ tr 'aeiou' 'AEIOU' < packages1.txt > packages1.trfile.txt
```

```
$ ls -l packages1.txt packages1.trfile.txt
```

```
$ cat packages1.trfile.txt
```

## Задание 2.2. Утилита `awk` примеры для изучения

Изучите примеры задания 2.2 и выполните их в ОС Ubuntu и Raspberry PI. **Выполнить примеры 1-4 в Raspberry PI, подключиться по ssh к Ubuntu и выполнить примеры 5-14.**

Включите ведение протокола командой `script` с журналом меток времени. **Протокол** назвать по следующему шаблону — `taskXФамилияNM`, где `X` — номер выполняемого задания, `Фамилия` — заменить на вашу фамилию латиницей и строчными буквами, `N` — номер группы, например 12 или 13, `M` — `r` — для Raspberry PI, `u` — для Ubuntu. **Журнал меток** назвать по следующему шаблону — `timelogXФамилияNM`, где `X` — номер выполняемого задания, `Фамилия` — заменить на вашу фамилию латиницей и строчными буквами, `N` — номер группы, например 12 или 13, `M` — `r` — для Raspberry PI, `u` — для Ubuntu.

### Примеры к заданию 2.2

#### **Пример 1.**

Предварительно создадим файл для обработки. Для этого выполним следующие действия:

1. Создаем каталог `examples` в домашнем каталоге пользователя.  

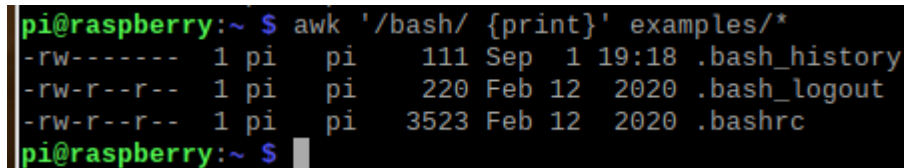
```
$ mkdir examples
```
2. Убедитесь, что Вы находитесь в домашнем каталоге, выполнив команду  

```
$ pwd
```
3. Перенаправим вывод команды `ls` расширенном формате в файл, например `log.txt`  

```
$ ls -la > examples/log.txt
```

4. И выполним команду для поиска данных в созданном файле или наборе файлов. Будем искать строки, содержащие слово «bash»

```
$ awk '/bash/ {print}' examples/*
```



```
pi@raspberrypi:~$ awk '/bash/ {print}' examples/*
-rw----- 1 pi pi 111 Sep 1 19:18 .bash_history
-rw-r--r-- 1 pi pi 220 Feb 12 2020 .bash_logout
-rw-r--r-- 1 pi pi 3523 Feb 12 2020 .bashrc
pi@raspberrypi:~$
```

Ищутся все файлы в каталоге **examples**, который находится в домашнем каталоге пользователя и в которых встречается слово **bash**. Результаты направляются на стандартное устройство вывода — действие **{print}**. В результатах выводятся строки, содержащие шаблон.

Действие направления на стандартное устройство вывода **{print}** обычно задается как действие по умолчанию, поэтому команду примера можно переписать так:

```
$ awk '/config/' examples/*
```

Результат при этом не изменится.

### Пример 2.

Найти по указанному пути файлы исходных текстов на языке C, включающие слова, например, **QUIT** или **SETDATE**:

```
$ awk '/QUIT|SETDATE/' /usr/include/protocols/*
```

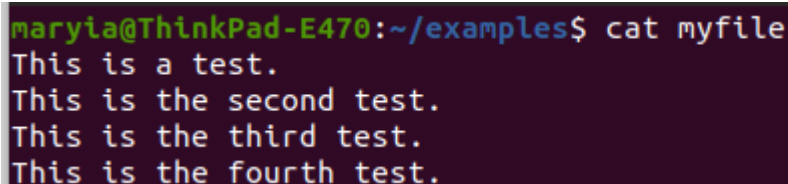
### Пример 3.

1. Создайте в каталоге **examples** текстовый файл **myfile** следующего содержания:

```
This is a test.
This is the second test.
This is the third test.
This is the fourth test.
```

2. Просмотрите файл:

```
$ cat myfile
```



```
maryia@ThinkPad-E470:~/examples$ cat myfile
This is a test.
This is the second test.
This is the third test.
This is the fourth test.
```

Одна из основных функций **awk** заключается в возможности манипулировать данными в текстовых файлах. Делается это путём автоматического назначения переменной каждому элементу в строке. По

умолчанию `awk` назначает следующие переменные каждому полю данных, обнаруженному им в записи:

- `$0` — представляет всю строку текста (запись).
- `$1` — первое поле.
- `$2` — второе поле.
- `$n` — `n`-ное поле.

2. Выполните вывод первого элемента каждой строки:

```
$ awk '{print $1}' myfile
```

3. Выполните вывод третьего элемента каждой строки:

```
$ awk '{print $3}' myfile
```

#### **Пример 4.**

1. **awk** позволяет изменять регистр символов. В примере ниже необходимо ввести команду и две строки текста, одна из которых будет преобразована в строку в верхнем регистре:

```
$ awk '/foo/ { print toupper($0); }'
```

```
This line contains bar.
```

```
This line contains foo.
```

```
THIS LINE CONTAINS FOO.
```

2. Для завершения ввода нажмите комбинацию клавиш `Ctrl+C`.

#### **Пример 5.**

1. Создать в каталоге **examples** текстовый файл `list_students`, отсортированный по алфавиту, каждая строка которого включает: фамилию, имя, факультет, курс, рейтинговая оценка, и состоящий из следующих строк-записей:

```
Асташко Иван ПИ 1 6
```

```
Бузун Евгений КБ 1 9
```

```
Кравченя Леонид ПМ 1 4
```

```
Кошкин Владимир ИН 1 7
```

```
Липин Федор ПИ 2 4
```

```
Пенов Николай ПМ 1 5
```

```
Яшин Петр КБ 4 8
```

2. Вывести список, состоящий из фамилий, имен и рейтинговых оценок:

```
$ awk '{print $1,$2,$5}' list_students
```

### **Пример 6.**

Выбрать из исходного файла *list\_students* студентов с рейтинговой оценкой 8 и распечатать все значения полей для выбранных записей:

```
$ awk '/8/ {print $0}' list_students
```

В случае, если оценка встречается в других полях, например, курс 4 и оценка 4, то условия выборки недостаточны.

### **Пример 7.**

Выбрать из списка студентов, рейтинговая оценка которых 4:

```
$ awk '($5==4)' list_students
```

### **Пример 8.**

Выбрать из списка студентов специальности КБ, 1 курса:

```
$ awk '($3=="КБ")&&($4==1)' list_students
```

### **Пример 9.**

Выбрать из файла со списком студентов все имена с длиной >5

```
$ awk '(length($2)>5) {print}' list_students
```

### **Пример 10.**

Вывести данные исходного файла в виде таблицы и пронумеровать строки:

```
$ awk '{print NR, $0}' list_students
```

### **Пример 11.**

Вывести все имена студентов в верхнем регистре:

```
$ awk '{print toupper ($2)}' list_students
```

### **Пример 12.**

Посчитать суммарный балл оценок студентов:

```
awk '{sum += $5} END {print("SUM=",sum)}' list_students
```

### **Пример 13.**

Присутствует ли фамилия Леонов в списке файла *list\_students*?

```
$ awk '($1 ~ /Кравченя/)' list_students
```

### **Пример 14.**

Проверить, есть ли запись студента Липина в списке файла *list\_students*, причем фамилия может быть написана с прописной или строчной буквы:

```
$ awk '($1 ~ /[Лл]ипин/)' list_students
```

Вывести все записи кроме студента Липин:

```
$ awk '($1 ~ /[Лл]ипин/)' list_students
```

### Пример 15.

1. Создайте текстовый файл **colours.csv** вида:

**name,color,amount**

apple,red,4

banana,yellow,6

strawberry,red,3

grape,purple,10

apple,green,8

plum,purple,2

kiwi,brown,4

potato,brown,9

pineapple,yellow,5

2. Выберите все записи из файла **colours.csv**, количество которых больше 5, выполнив команду:

```
$ awk -F", " '$3>6 {print $1, $2}' colours.csv > output.txt
```

В команде выше явно был указан разделитель данных, т.е. “,”. В результате выполнения команды будет создан файл, содержащий записи согласно условию.

## Задание 2.3. Обработка строк файла с применением потокового редактора sed

Изучите примеры задания 2.3 и выполните их в ОС Ubuntu и Raspberry PI. **Выполнить примеры 1-4 в Raspberry PI, скопировать файл books в Ubuntu, подключиться по ssh к Ubuntu и выполнить примеры 5-8.**

Включите ведение протокола командой `script` с журналом меток времени. **Протокол** назвать по следующему шаблону — `taskXФамилияNM`, где `X` — номер выполняемого задания, `Фамилия` — заменить на вашу фамилию латиницей и строчными буквами, `N` — номер группы, например 12 или 13, `M` — `r` — для Raspberry PI, `u` — для Ubuntu. **Журнал меток** назвать по следующему шаблону — `timelogXФамилияNM`, где `X` — номер выполняемого задания, `Фамилия` — заменить на вашу фамилию латиницей и строчными буквами, `N` — номер группы, например 12 или 13, `M` — `r` — для Raspberry PI, `u` — для Ubuntu.

### Примеры к заданию 2.3

1. Сохраните данные ниже в файл с именем `books`:

Book one.

The second book.

The third.

This is book four.

Five.

This is the sixth.

This is book seven.

Eighth and last.

2. Чтобы вывести все строки файла и продублировать строки согласно шаблону, т. е. содержащие слово book выполните команду

```
$ sed '/book/ p' books
```

3. Чтобы выбрать определенные строки, например содержащие слово book, выполните команду

```
$ sed -n '/book/ p' books
```

4. Чтобы вывести часть файла, например, строки с 2 по 5, выполните команду

```
$ sed -n '2,5 p' books
```

5. Для выполнения более сложных и длинных инструкций можем использовать файл программы для sed. Выполним команду из примера 4, указав параметры в файле records.

Содержимое файла records:

```
2,5 p
```

Пример команды из примера 4 с выполнением инструкций из файла:

```
$ sed -n -f records books
```

6. В данном примере выбираем строку 3 и используем инструкцию Добавить, чтобы добавить разделитель строк и текст «My favorite book.» к третьей строке:

Содержимое файла appends

```
3 a\
```

```
My favorite book.
```

Пример команды с добавлением строки:

```
$ sed -f appends books
```

7. В данном примере требуется вставить разделитель строк и текст «SKARBONKA.» перед строками, в которых содержится слово «This»:

Содержимое файла insert

```
/This/ i\
```

```
SKARBONKA.
```

Пример команды с со вставкой строк:

```
$ sed -f insert books
```

8. В следующем примере выполняется замена слова book на novel

```
$ sed -n 's/book/novel/ p' books
```

## Задание 2.4. Исполнительная среда оболочки Bash

Изучите примеры задания 2.4 и выполните их в ОС Ubuntu и Raspberry PI.

Имена скриптов могут быть любыми.

Включите ведение протокола командой `script` с журналом меток времени. **Протокол** назвать по следующему шаблону — `taskXФамилияNM`, где `X` — номер выполняемого задания, `Фамилия` — заменить на вашу фамилию латиницей и строчными буквами, `N` — номер группы, например 12 или 13, `M` — `r` — для Raspberry PI, `u` — для Ubuntu. **Журнал меток** назвать по следующему шаблону — `timelogXФамилияNM`, где `X` — номер выполняемого задания, `Фамилия` — заменить на вашу фамилию латиницей и строчными буквами, `N` — номер группы, например 12 или 13, `M` — `r` — для Raspberry PI, `u` — для Ubuntu.

### 2.4.1. Примеры для изучения: Исполнительная среда оболочки

**Выполнить примеры 1-3 в Raspberry PI, подключиться по ssh к Ubuntu и выполнить примеры 4-12.**

1. Войдите в систему под своей учетной записью.

2. Выполните следующие команды, которые используют переменные `a`, `full_name` и `short_name` командной оболочки Bash.

```
$ a=879
```

```
$ echo "The value of \"a\" is $a."
```

```
$ export full_name="John Lee"
```

```
$ echo $full_name
```

```
$ export short_name="John"
```

```
$ export short_name
```

```
$ echo $ short_name
```

3. Выполните следующие команды, которые иллюстрируют операции над переменными

```
$ export foo=""
```

```
$ echo ${foo:-one}
$ echo $foo
$ echo ${foo:=one}
$ export foo="this is a test"
$ echo $foo
$ echo ${foo:+bar}
```

4. Выполнить следующие команды, которые создают переменные массивы.

```
$ ARR[1]=one
$ ARR[2]=two
$ ARR[3]=three
$ ARR1=(zero one two three)
```

5. Выполнить следующую команду, которая выводит на консоль значения элементов массива.

```
$ echo ${ARR[0]} ${ARR[1]}
```

6. Выполните следующие команды для удаления элемента ARR[1] массива ARR и для удаления всего массива ARR:

```
$ unset ARR[1]
$ unset ARR
```

7. Создайте псевдоним для команды clear, выполнив следующую команду:

```
$ alias c='clear'
```

Просмотрите созданные псевдонимы.

```
$ alias
$ c
```

8. Этот псевдоним будет утерян, если вы выйдете из командной оболочки, а затем войдете в нее вновь. Чтобы обеспечить сохранность псевдонима при каждом входе в командную оболочку пользователя student, нужно выполнить следующие действия. Откройте файл .bashrc в текстовом редакторе, например vim

```
$ vim .bashrc
```

Найдите в файле bashrc строку, которая содержит текст:

```
# User specific aliases and functions.
```

Добавьте после этой строки следующую строку:

```
alias c='clear'
```

Сохраните файл и выйдите из текстового редактора.

9. Чтобы проверить сохранение псевдонима, сначала выйдите из командной оболочки, а затем снова войдите в неё под именем student и выполните команду

```
$ alias
$ c
```

10. Отобразите в терминале текущее значение вашей строки приглашения, выполнив команду

```
$ echo $PS1
```

11. Измените вашу строку приглашения, выполнив следующую команду:

```
$ PS1='Linux -> '
```



12. Восстановите традиционную строку приглашения, выполнив следующую команду:

```
$ PS!='$\h $ '
```

## 2.4.2. Примеры для изучения: Исполнение команд

**Выполнить примеры 1-3 в Raspberry PI, подключиться по ssh к Ubuntu и выполнить примеры 4-7.**

1. Выполните следующую команду, которая иллюстрирует расширение командным интерпретатором фигурных скобок.

```
$ echo sp{el,il,al}l  
spell spill spall
```

2. Выполните следующую команду, которая иллюстрирует расширение командным интерпретатором символа тильда на домашний каталог пользователя.

```
$ cat ~/message.txt
```

3. Выполните следующую команду, которая иллюстрирует расширение командным интерпретатором переменной.

```
$ echo $SHELL  
/bin/bash
```

4. Выполните следующую команду, которая иллюстрирует расширение командным интерпретатором команд.

```
$ echo $(date)
```

5. Объявите переменные X и Y и задайте им некоторые значения. Выполните следующую команду, которая иллюстрирует расширение командным интерпретатором арифметического выражения.

```
$ echo Area: $[X * Y]
```

6. Выполните следующую команду, которая иллюстрирует запрещение расширения символа, который следует за символом обратный слэш.

```
$ echo Your cost: \$5,00
```

7. Выполните следующую команду, которая иллюстрирует запрещение расширения символов, заключенных в одинарные кавычки.

```
$ echo '$date'  
$ date
```

## 2.4.3. Примеры для изучения: Сценарии Bash

**Выполнить примеры 1-2 в Raspberry PI, подключиться по ssh к Ubuntu и выполнить примеры 3-12.**

### Пример 1.

1. Создайте текстовый файл *ascript.sh*, который содержит следующий текст.

```
#!/bin/bash  
# This script displays some information about your
```

```
environment
echo "Hello."
echo "The date and time are $(date)"
echo "Your working directory is: $(pwd)"
2. Сохраните файл в текущем каталоге.
3. Сделайте файл ascript.sh исполняемым, выполнив команду
# chmod u+x ascript.sh
4. Запустите сценарий из командной строки, выполнив команду
# ./ascript.sh
```

### Пример 2.

1. Создайте и выполните следующий скрипт, который поясняет использование команды `read` для ввода данных с терминала.

```
#!/bin/bash
echo -n "Enter your name and press [ENTER]: "
read var_name
echo "Your name is: $var_name"
```

2. Выполните следующие команды, которые иллюстрируют использование команды `printf` для форматированного вывода данных.

```
# printf "%d\n" 5
# printf "Hello, $USER.\n\n"
# distance=15
# printf "Distance is %5d Miles\n" $distance
# printf "%d\n" 0xF
# printf "0x%X\n " 15
```

### Пример 3.

При разработке скриптов можем создавать скрипты не только на языке `bash`, но и использовать инструкции `awk`. В пример рассматривается пример создания сценария `awk`.

При выполнении данного примера используйте файл ***colours.csv*** из в примера 15 задания 2.2.

1. Файл, содержащий `awk`-инструкции, может быть преобразован в файл сценария. Для этого создайте файл, например ***example.awk*** вида:

```
#!/usr/bin/awk -f
#
# Вывод строк с нумерацией без строки с заголовками.
#

NR > 1 {
printf "%d: %s\n", NR, $0
}
```

2. Назначьте права на выполнение:  
`$ chmod u+x example.awk`
3. Выполните скрипт:  
`$ ./example.awk colours.csv`
4. В результате будет выведен список пронумерованных строк.

#### **Пример 4.**

1. Создайте и выполните следующий скрипт, который иллюстрирует использование условной инструкции if-then-else, сравнивая значения двух переменных.

```
#!/bin/bash
V1="foo"
V2="bar"
if [ "$V1" = "$V2" ]; then
    echo true
else
    echo false
fi
```

#### **Пример 5.**

1. Создайте и выполните следующий скрипт, который объясняет использование условной инструкции case для выбора нужного варианта значения переменной.

```
#!/bin/bash
printf 'Which Linux distribution do you know? '
read DISTR
case $DISTR in ubuntu)
    echo "I know ubuntu." ;;
centos|rhel)
    echo " I know it too." ;;
windows)
    echo "Very funny." ;;
*)
    echo "I don't know it." ;;
esac
```

#### **Пример 6.**

- Создайте и выполните следующий скрипт, который иллюстрирует использование инструкции цикла for для числовых и символьных значений, а также показывает, как программируются бесконечные циклы for.

```
#!/bin/bash
for (( c=1; c<=3; c++ ))
do
    echo "$c iteration"
done
```

```
# Loop through a set of strings:
for m in Apple Sony Panasonic "Hewlett Packard" Nokia
do
    echo "Manufacturer is:" $m
done
```

```
for (( ; ; ))
do
    echo "infinite loops [hit CTRL+C to stop]"
done
```

### **Пример 7.**

Создайте и выполните следующий скрипт, который иллюстрирует использование инструкций циклов `while` и `until`.

```
#!/bin/bash
x=1
while [ $x -le 3 ] do
    echo "x = $x"
    x=$(( $x + 1 ))
done
x=5 until [ $x -le 3 ]
do
    echo "x = $x"
    x=$(( $x - 1 ))
done
```

### **Пример 8.**

Создайте и выполните следующий скрипт, который иллюстрирует использование команды принудительного выхода из цикла `continue`.

```
#!/bin/bash
for myloop in 1 2 3 4 5
do
    if [ "$myloop" -eq 3 ] then
        continue
    # Skip rest of this particular loop iteration.
    fi
    echo -n "$myloop"
done
```

### **Пример 9.**

Создайте и выполните следующий скрипт, который иллюстрирует использование команды принудительного завершения цикла `break`.

```
#!/bin/bash
for myloop in 1 2 3 4 5
```

```
do
    echo -n "$myloop"
if [ "$myloop" -eq 3 ] then
    break
    # This line will break out of the loop
fi
done
```

### **Пример 10.**

Создайте и выполните следующий скрипт, который выводит свое имя, количество аргументов и значения первых трех аргументов, иллюстрируя тем самым обработку аргументов, которые передаются скрипту при вызове. При вызове этого скрипта передайте ему три произвольных аргумента.

```
#!/bin/bash
echo "My name: $0"
echo "Total number of arguments: $# "
echo "Argument 1: $1"
echo "Argument 2: $2"
echo "Argument 3: $3"
```

### **Пример 11.**

Создайте и выполните следующий скрипт, который выводит все свои аргументы, иллюстрируя тем самым обработку произвольного количества аргументов. Вызовите этот скрипт, передав ему произвольное количество аргументов.

```
#!/bin/bash
numargs=$#
for ((i=1 ; i <= numargs ; i++)) do
    echo "$1"
    shift
done
```

### **Пример 12.**

В командной строке объявите и вызовите функцию mcd, которая создает каталог и делает его текущим.

```
$ function mcd() { mkdir $1 && cd $1; }
$ mcd temp
```

## **Задание 2.5. Для самостоятельной работы**

Во всех заданиях должен быть контроль ошибок (если к какому-либо каталогу нет доступа, необходимо вывести соответствующее сообщение и продолжить выполнение). Вывод сообщений об ошибках должен производиться в стандартный поток вывода сообщений об ошибках (stderr) в следующем виде:

Имя\_модуля: текст\_сообщения.

Пример: ./script1 : error open file: 1.txt.

Имя модуля, имя файла берутся из аргументов командной строки.

В качестве ответа предоставить коды скриптов в репозитории и логи команды script, включая метки времени. **Протокол** назвать по следующему шаблону — taskXФамилияNM, где X — номер выполняемого задания, Фамилия — заменить на вашу фамилию латиницей и строчными буквами, N — номер группы, например 12 или 13, M — r — для Raspberry PI, u — для Ubuntu. **Журнал меток** назвать по следующему шаблону — timelogXФамилияNM, где X — номер выполняемого задания, Фамилия — заменить на вашу фамилию латиницей и строчными буквами, N — номер группы, например 12 или 13, M — r — для Raspberry PI, u — для Ubuntu.

## Варианты

Для всех вариантов задач 4-5 проверить на Raspberry PI и подключившись по ssh проверить в Ubuntu, запустив их на выполнение в терминале, созданном мультиплексором терминала screen.

Для задач 4-5 проверить отложенный запуск скрипта, отсоединиться от терминала, созданного screen, подключиться к терминалу и проверить результаты работы скрипта.

Номер варианта индивидуального задания K равен числу букв N1 вашей фамилии, умноженному на число букв N2 Вашего имени (по паспорту), умноженному на число букв N3 вашего отчества по модулю 7:  $k = (N1 * N2 * N3) \bmod 7$

## Вариант 0.

Задача 1. Перенаправьте вывод команды ls, просматривающей содержимое текущего каталога в файл file\_list.txt. В выводе ls отображать все файлы, включая скрытые, inode (индексный дескриптор) файла и размер в килобайтах. Просмотрите содержимое файла file\_list.txt.

Задание 2. Напишите AWK-программу для вывода четных строк списка автомобилей из файла cars.txt (<https://edufpmi.bsu.by/mod/resource/view.php?id=464>), в котором буквами в верхнем регистре будет обозначен только Модель, а производитель с первой заглавной буквы. Последующие поля каждой строки должны появляться в том виде, в котором они сохранены в файле cars.

Задание 3. Напишите sed-команду, копирующую файл на стандартный

вывод, удаляя при этом все строки, начинающиеся со слова «Yesterday» или «TODAY» в указанном написании. Проверить для случаев: а) наличие слов в начале строки, середине и в конце и в разных регистрах.

Задача 4. Напишите скрипт, который создает файл с заданным именем и устанавливает для него заданную маску разрешений на доступ. После завершения работы скрипта, маска разрешений на доступ должна быть установлена в значение, используемое по умолчанию. Использовать команду `umask`.

Задача 5. Написать скрипт для поиска файлов заданного размера в заданном каталоге и во всех его подкаталогах (имя каталога задаётся пользователем в качестве третьего аргумента командной строки). Диапазон (мин.– макс.) размеров файлов задаётся пользователем в качестве первого и второго аргумента командной строки. Проверить работу программы для каталога `/usr` и диапазона (мин.– макс.) `1000 1510`.

### **Вариант 1.**

Задача 1. Создайте в текущем каталоге новый каталог `docs`. Для команды `mkdir` использовать атрибут для вывода сообщения о созданном каталоге и задать права доступа, равные `751`. Вывод команды `mkdir` направьте в файл `mkdir_log.txt`. Просмотрите содержимое файла `mkdir_log.txt`.

Задание 2. Напишите AWK-программу, которая использует файл `cars.txt` (<https://edufpmi.bsu.by/mod/resource/view.php?id=464>) выводит данные о всех автомобилях, цена которых меньше \$6000, с названием модели в верхнем регистре и отправляет эти данные на стандартный вывод, а также подсчитывает суммарное значение стоимости для выведенных и выводит последней строкой.

Задание 3. Напишите `sed`-программу по имени `ins`, которая копирует файл на стандартный вывод, заменяя все встречающиеся слова `cat` на `dog` и предваряя каждую измененную строку строкой предупреждения «*Следующая строка была изменена:*» .

Задача 4. Напишите скрипт, который создает файл с заданным именем и устанавливает для него заданную маску разрешений на доступ. После завершения работы скрипта, маска разрешений на доступ должна быть установлена в значение, используемое по умолчанию.

Задача 5. Написать скрипт с использованием цикла `for`, выводящий на

консоль размеры и права доступа для всех файлов в заданном каталоге и во всех его подкаталогах (имя каталога задается пользователем в качестве первого аргумента командной строки). Проверить работу программы для каталога /usr.

### **Вариант 2.**

Задача 1. Создайте псевдоним `lr`, который вызывает команду `ls` со следующими возможностями: псевдоним перечисляет файлы в длинном формате. Перенаправьте вывод команды `lr` в текущем каталоге в файл `file_list.txt`. Просмотрите содержимое файла `file_list.txt`.

Задание 2. Напишите AWK-программу, которая использует файл `cars.txt` (<https://edufpmi.bsu.by/mod/resource/view.php?id=464>) выводит данные о всех автомобилях, цена которых превышает \$5000, и отправляет эти данные на стандартный вывод, а также подсчитывает среднее значение стоимости для выведенных и выводит последней строкой.

Задание 3. Напишите `sed` -команду, копирующую на стандартный вывод только те строки файла, которые начинаются со слова `Today`.

Задача 4. Напишите скрипт, который запрашивает номер зачетной книжки и переводите его в восьмеричную систему счисления и устанавливает для существующего файла заданную маску разрешений на доступ. После завершения работы скрипта, маска разрешений на доступ должна быть установлена в значение, используемое по умолчанию.

Задача 5. Написать скрипт для поиска заданной пользователем строки во всех файлах заданного каталога и во всех его подкаталогах (строка и имя каталога задаются пользователем в качестве первого и второго аргумента командной строки). На консоль выводятся полный путь и имена файлов, в содержимом которых присутствует заданная строка, и их размер. Если к какому-либо каталогу нет доступа, необходимо вывести соответствующее сообщение и продолжить выполнение. Проверить работу программы для каталога /usr и строки «`stdio.h`».

### **Вариант 3.**

Задача 1. Используя команду `who`, определите пользователей, работающих в системе и отобразите данные о дате и времени загрузки системы, текущий уровень, на котором выполняются процессы пользователя, и сведения о пользователе. Вывод команды `who` направьте в файл `users.txt`. Просмотрите



содержимое файла users.txt.

Задание 2. Используйте AWK для определения, сколько строк в файле /usr/share/dict/words ([проверить на Ubuntu](#)) содержат подстроку `abul`, а так же выводите их нумеруя. Определите самую длинную строку. Если данный файл отсутствует выполнять задание для любого текстового файла, который можно найти в подкаталогах каталога /usr/share.

Задание 3. Напишите sed-программу по имени `div`, которая копирует файл из 50 строк на стандартный вывод, причем копирует строки 5-10 в файл по имени `first` и копирует последние 12 строк в файл по имени `last`.

Задача 4. Напишите скрипт, который выполняет следующие действия:

- Выводит на терминал меню, которое предлагает выбор из следующих действий:
  - удалить файл;
  - переименовать файл;
  - переместить файл;
  - создать файл.
- запрашивает имена файлов, для выбранного действия; выполняет выбранное действие.

Задача 5. Написать скрипт поиска одинаковых по их содержимому файлов в двух каталогах, например `Dir1` и `Dir2`. Пользователь задаёт имена `Dir1` и `Dir2` в качестве первого и второго аргумента командной строки. В результате работы программы файлы, имеющиеся в `Dir1`, сравниваются с файлами в `Dir2` по их содержимому. На экран выводятся число просмотренных файлов и результаты сравнения. Проверить работу программы для каталога /usr (`Dir1`) и любого каталога в каталоге /home (`Dir2`).

#### **Вариант 4.**

Задача 1. Создайте псевдоним `lr`, который вызывает команду `ls` со следующими возможностями: псевдоним классифицирует файлы, присоединяя индикатор типа файла к именам файлов. Перенаправьте вывод команды `lr` в текущем каталоге в файл `file_list.txt`. Просмотрите содержимое файла `file_list.txt`.

Задание 2. Напишите awk-программу, которая читает данные из файла <https://www.rfc-editor.org/rfc/rfc-ref.txt>, выводит строки с фразой «Network Protocol» подсчитывает их общее количество.

Задание 3. Сохраните вывод команды `ls -la` для каталога /etc в файл.

Напишите скрипт `sed` или `sed`-команду, выполняющую замену «`hosts`» на «`HOSTS`» и выводящую на экран все строки за исключением строк, начинающихся с символа «`d`».

Задача 4. Напишите сценарий, проверяющий имя текущего каталога и выводящий сообщение об ошибке, если оно короче пяти символов. Если больше или равно пяти символам, то выводящий посекундно в цикле имена файлов текущего каталога и их порядковый номер.

Задача 5.

Написать скрипт, находящий в заданном каталоге и во всех его подкаталогах все файлы, владельцем которых является заданный пользователь. Имя владельца и каталог задаются пользователем в качестве первого и второго аргумента командной строки. Скрипт выводит результаты в файл (третий аргумент командной строки) в виде полный путь, имя файла, его размер. На консоль выводится общее число просмотренных файлов. Проверить работу программы для каталога `/usr` пользователь `root`.

### **Вариант 5.**

Задача 1. Перенаправьте вывод списка файлов в длинном формате (расширенный формат) с сортировкой по размеру в файл `list.txt` (если файл существует, то новые строки в него должны быть добавлены) и вывести на стандартный вывод строки 4 первые строки из файла `list.txt`.

Задание 2. Напишите `awk`-программу, которая читает данные из файла <https://www.rfc-editor.org/rfc/rfc-retrieval.txt> и выводит последнее слово каждой строки с буквами, переведенными в верхний регистр.

Задание 3. Напишите `sed`-команду, копирующую файл `cars.txt` (<https://edufpmi.bsu.by/mod/resource/view.php?id=464>) на стандартный вывод, удаляя при этом все строки, содержащие «`ford`», и завершая вывод строкой с суммой стоимости всех моделей в выводе.

Задача 4. Требуется проверить, является ли файл обычным или он является каталогом. Если это обычный файл, то сценарий должен выводить имя файла и его размер. В случае, если размер файла превышает килобайт, то размер должен выводиться в килобайтах. Если размер превышает мегабайт — в мегабайтах.

Задача 5. Написать скрипт, находящий в заданном каталоге и во всех его

подкаталогах все файлы, заданного размера в заданном каталоге (имя каталога задаётся пользователем в качестве первого аргумента командной строки). Диапазон (мин.– макс.) размеров файлов задаётся пользователем в качестве второго и третьего аргументов командной строки. Скрипт выводит результаты поиска в файл (четвертый аргумент командной строки) в виде: полный путь, имя файла, его размер. На консоль выводится общее число просмотренных файлов. Проверить работу программы для каталога /usr и диапазона (мин.– макс.) 1000 1010.

### **Вариант 6.**

Задача 1. Создайте псевдоним `lr`, который вызывает команду `ls` со следующими возможностями: псевдоним перечисляет файлы в порядке времени их модификации. Перенаправьте вывод команды `lr` в текущем каталоге в файл `file_list.txt`. Просмотрите последние 7 строк содержимого файла `file_list.txt`.

Задание 2. Напишите AWK-программу для вывода списка автомобилей из файла `cars.txt` (<https://edufpmi.bsu.by/mod/resource/view.php?id=464>), которая вывод самый старый и самый новый автомобиль chevy с выводом модели заглавными буквами.

Задание 3. Напишите sed-команду, копирующую файл на стандартный вывод, удаляя при этом все строки, начинающиеся с символа `#` или цифры.

Задача 4. Напишите сценарий, который генерирует тысячу файлов `1.txt` .... `1000.txt`, и в каждый файл записывает подряд 100 чисел `N`, где `N` = порядковый номер файла. Затем скрипт должен соединить в один файл все файлы с четными номерами (`even.txt`) и в другой файл — все файлы с нечетными номерами (`odd.txt`).

Задача 5.

Написать скрипт, подсчитывающий суммарный размер файлов в заданном каталоге и во всех его подкаталогах (имя каталога задаётся пользователем в качестве аргумента командной строки). Скрипт выводит результаты подсчёта в файл (второй аргумент командной строки) в виде: каталог (полный путь), суммарный размер файлов, число просмотренных файлов. Проверить работу программы для каталога /usr.