Отчёт по лабораторной работе №11

Дисциплина: Операционные системы

Аветисян Давид Артурович

Содержание

Список таблиц

Список иллюстраций

1 Цель работы

Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы.

2 Задание

1. Написать скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. Способ использования команд архивации необходимо узнать, изучив справку.
2. Написать пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов.
3. Написать командный файл — аналог команды ls (без использования самой этой команды и команды dir). Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога.
4. Написать командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки.

3 Выполнение лабораторной работы

1. Для начала я изучил команды архивации, используя команды «man zip», «man bzip2», «man tar» (рис. -fig. 3.1).

```
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab11$ man zip
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab11$ man bzip2
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab11$ man tar
```

Рис. 3.1: Изучил команды архивации

Синтаксис команды zip для архивации файла (рис. -fig. 3.2):
zip [опции] [имя файла.zip] [файлы или папки, которые будем архивировать]
Синтаксис команды zip для разархивации/распаковки файла:
unzip [опции] [файл_архива.zip] [файлы] -x [исключить] -d [папка]

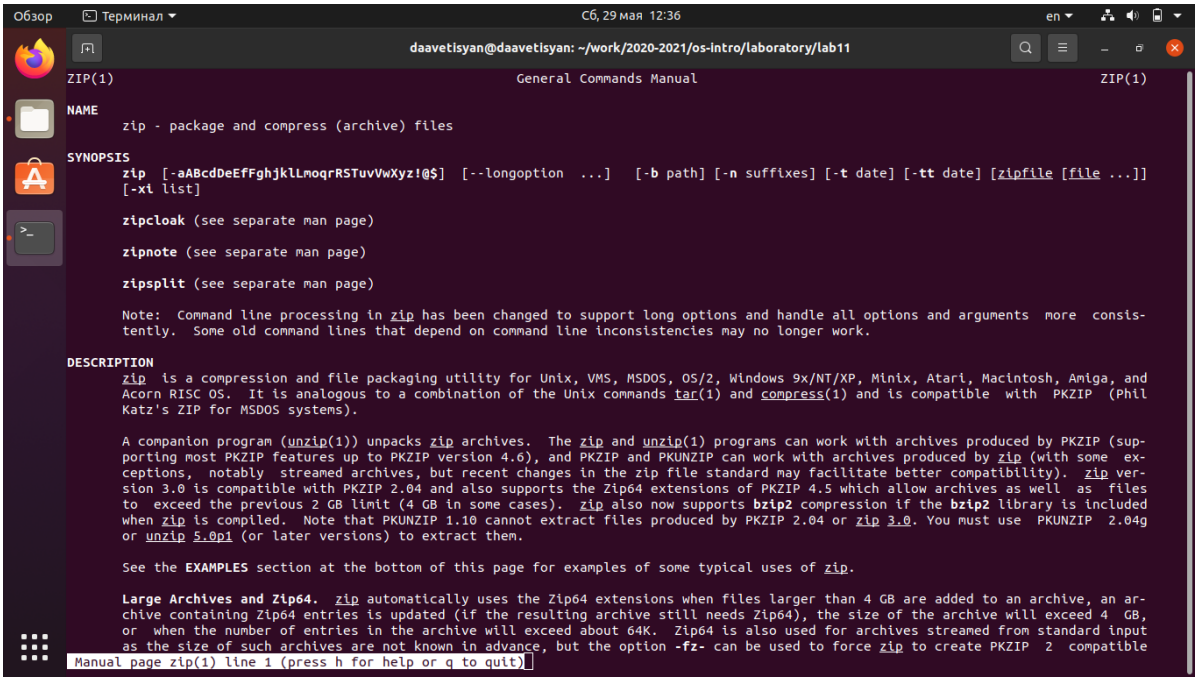

Рис. 3.2: Синтаксис команды zip

Синтаксис команды bzip2 для архивации файла (рис. -fig. 3.3):

bzip2 [опции] [имена файлов]

Синтаксис команды bzip2 для разархивации/распаковки файла:

bunzip2 [опции] [архивы.bz2]

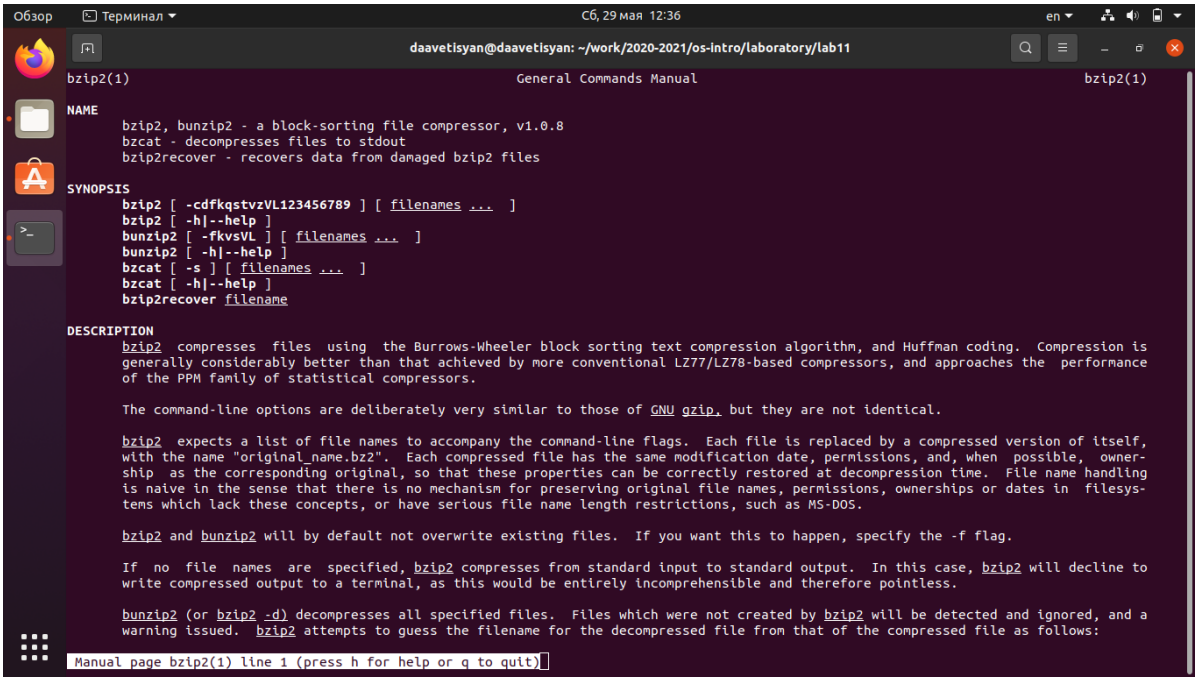

Рис. 3.3: Синтаксис команды bzip2

Синтаксис команды tar для архивации файла (рис. -fig. 3.4):

tar [опции] [архив.tar] [файлы_для_архивации]

Синтаксис команды tar для разархивации/распаковки файла:

tar [опции] [архив.tar]

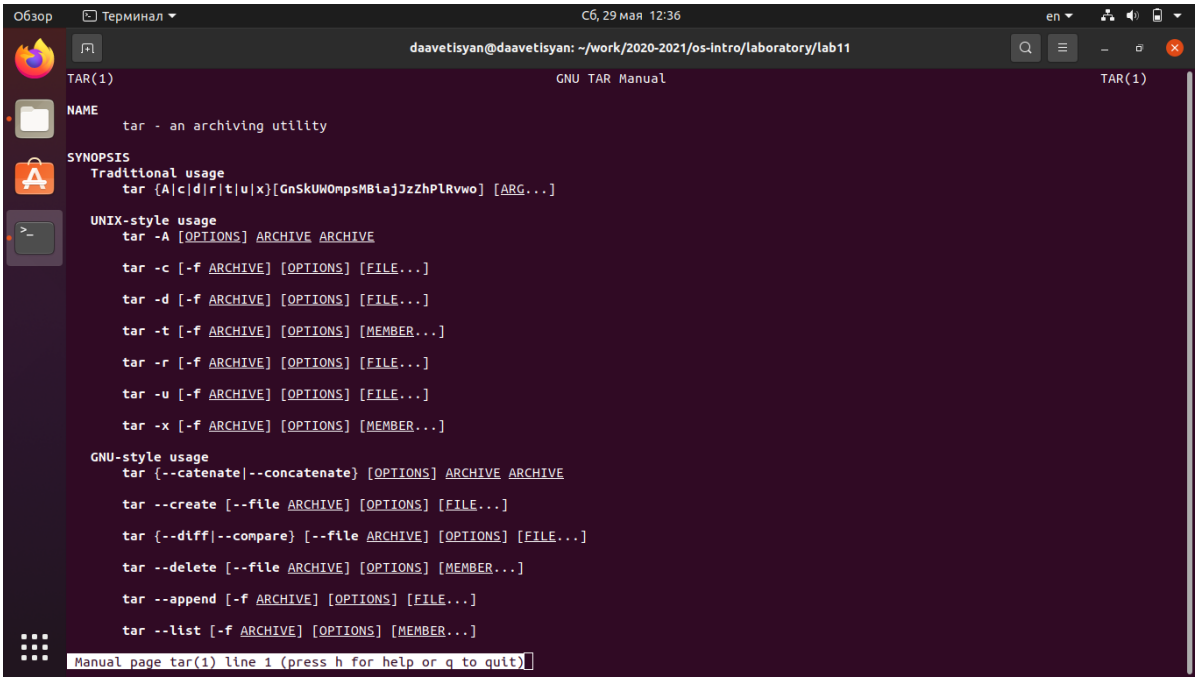

Рис. 3.4: Синтаксис команды tar

Далее я создал файл, в котором буду писать первый скрипт, и открыл его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch backup.sh» и «emacs &») (рис. -fig. 3.5).

```
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab11$ touch backup.sh
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab11$ emacs &
```

Рис. 3.5: Создал файл для первого скрипта

После написал скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в моём домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar (рис. -fig. 3.6). При написании скрипта использовал архиватор bzip2.

```
#!/bin/bash

name="backup.sh"
mkdir ~/backup
bzip2 -k ${name}
mv ${name}.bz2 ~/backup/
echo "Выполнено"
```

Рис. 3.6: Написал первый скрипт

Проверил работу скрипта (команда «./backup.sh»), предварительно добавив для него право на выполнение (команда «chmod +x *.sh»). Проверил, появился ли каталог backup/, перейдя в него (команда «cd backup/»), посмотрел его содержимое (команда «ls») и просмотрел содержимое архива (команда «bunzip2 -c backup.sh.bz2») (рис. -fig. 3.7) (рис. -fig. 3.8). Скрипт работает корректно.

```
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab11$ chmod +x *.sh
[1]+  Завершён        emacs
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab11$ ls -l
итого 20
-rwxrwxr-x 1 daavetisyan daavetisyan  113 мая 29 12:44 backup.sh
-rwxrwxr-x 1 daavetisyan daavetisyan  113 мая 29 12:43 backup.sh~
-rwxrwxr-x 1 daavetisyan daavetisyan  131 мая 29 12:43 backup.sh.bz2
drwxrwxr-x 3 daavetisyan daavetisyan 4096 мая 29 12:11 pres11
drwxrwxr-x 3 daavetisyan daavetisyan 4096 мая 29 12:10 report11
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab11$ ls -l
итого 16
-rwxrwxr-x 1 daavetisyan daavetisyan  113 мая 29 12:44 backup.sh
-rwxrwxr-x 1 daavetisyan daavetisyan  113 мая 29 12:43 backup.sh~
drwxrwxr-x 3 daavetisyan daavetisyan 4096 мая 29 12:11 pres11
drwxrwxr-x 3 daavetisyan daavetisyan 4096 мая 29 12:10 report11
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab11$ ./backup.sh
Выполнено
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab11$ cd ~/backup
daavetisyan@daavetisyan:~/backup$ ls
backup.sh.bz2
daavetisyan@daavetisyan:~/backup$ import report11/image11/img07.png
```

Рис. 3.7: Проверил первый скрипт

```
daavetisyan@daavetisyan:~/backup$ bunzip2 -c backup.sh.bz2
#!/bin/bash

name="backup.sh"
mkdir ~/backup
bzip2 -k ${name}
mv ${name}.bz2 ~/backup/
echo "Выполнено"
```

Рис. 3.8: Проверил первый скрипт

2. Создал файл, в котором буду писать второй скрипт, и открыл его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch prog2.sh» и «emacs &») (рис. -fig. 3.9).

```
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab11$ touch prog2.sh
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab11$ emacs &
```

Рис. 3.9: Создал файл для второго скрипта

Написал пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов (рис. -fig. 3.10).

```
#!/bin/bash

echo "Аргументы"
for a in $@
do echo $a
done
```

Рис. 3.10: Написал второй скрипт

Проверил работу написанного скрипта (команды «./prog2.sh 1 2 3 4 5» и «./prog2.sh 1 2 3 4 5 6 7 8 9 10 11 12»), предварительно добавив для него право на

выполнение (команда «chmod +x *.sh»). Вводил аргументы, количество которых меньше 10 и больше 10 (рис. -fig. 3.11). Скрипт работает корректно.

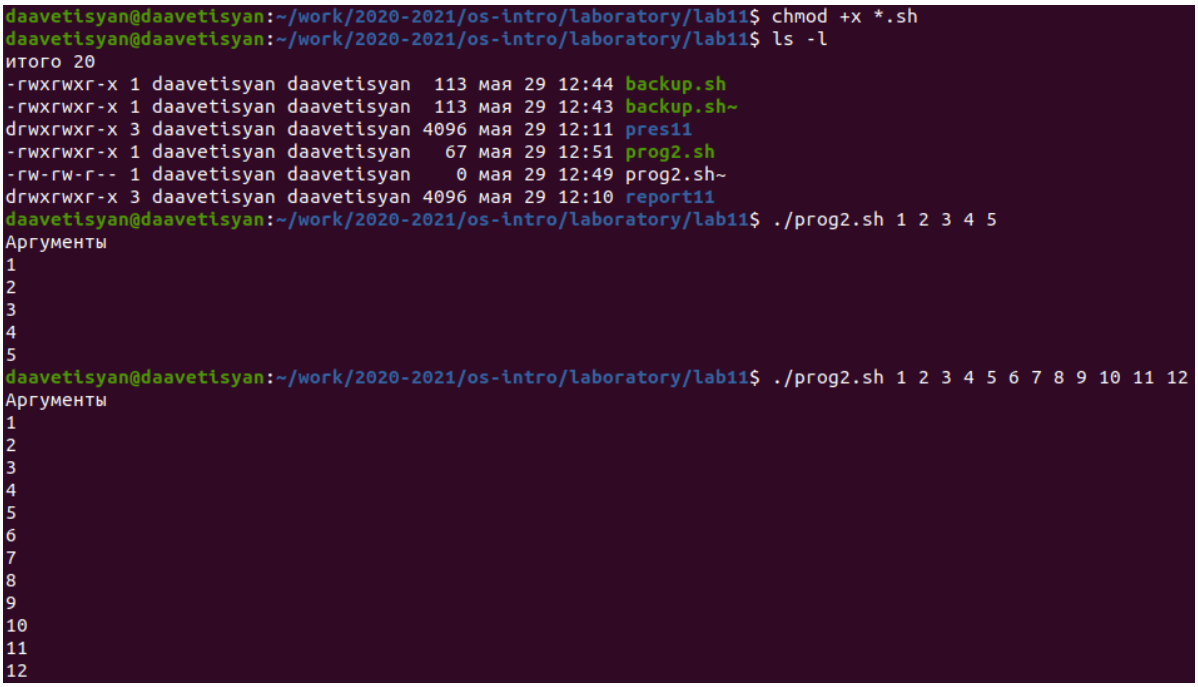

Рис. 3.11: Проверил второй скрипт

3. Создал файл, в котором буду писать третий скрипт, и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch ls.sh» и «emacs &») (рис. -fig. 3.12).

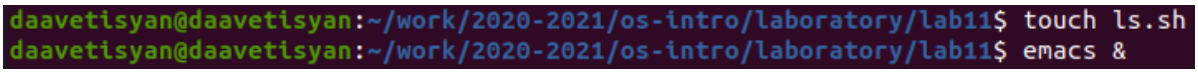

Рис. 3.12: Создал файл для третьего скрипта

Написал командный файл – аналог команды ls (без использования самой этой команды и команды dir). Он должен выдавать информациюо нужном каталоге и выводить информацию о возможностях доступа к файлам этого каталога (рис. -fig. 3.13).

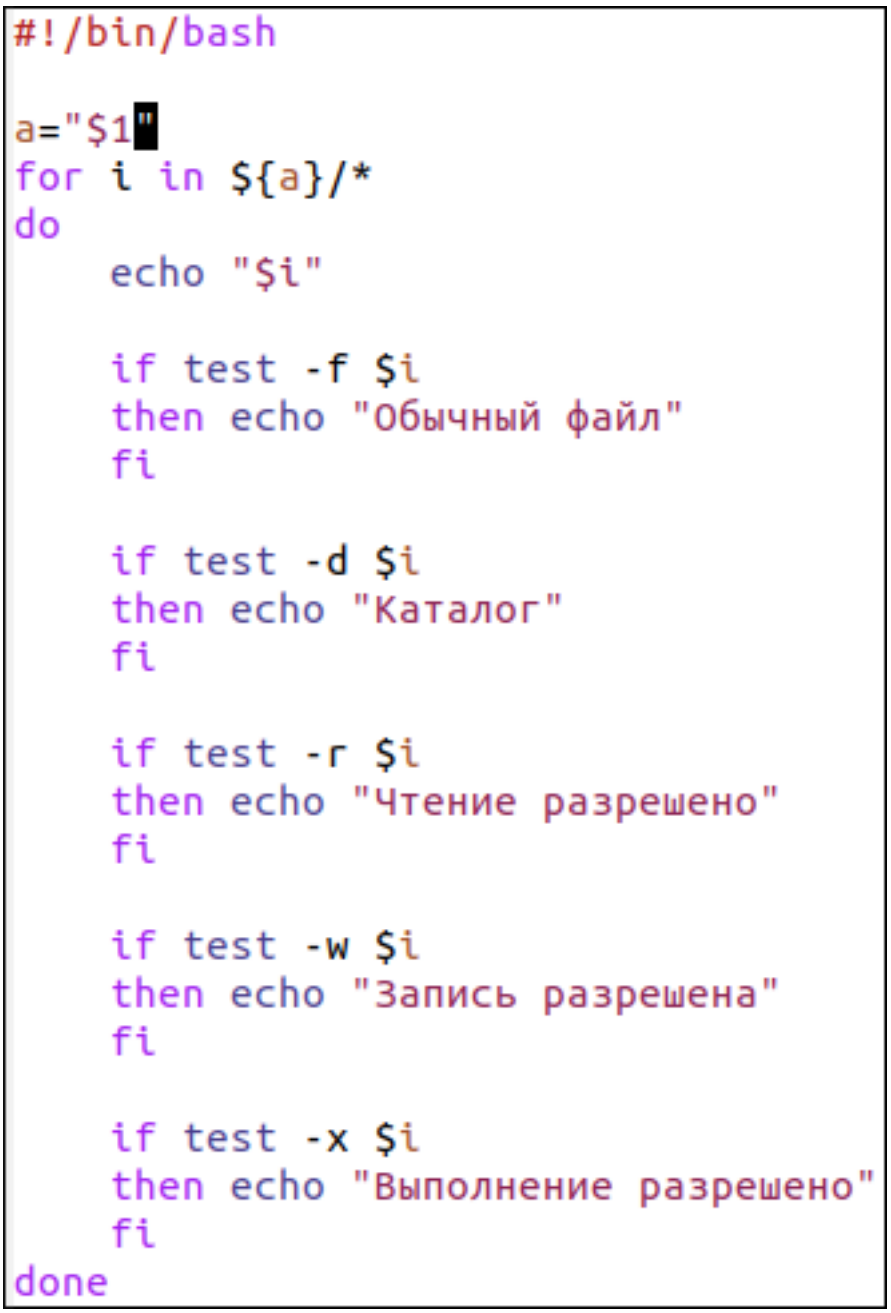

Рис. 3.13: Написал третий скрипт

Далее проверил работу скрипта (команда «./progls.sh ~»), предварительно добавив для него право на выполнение (команда «chmod +x *.sh») (рис. -fig. 3.14). Скрипт работает корректно.

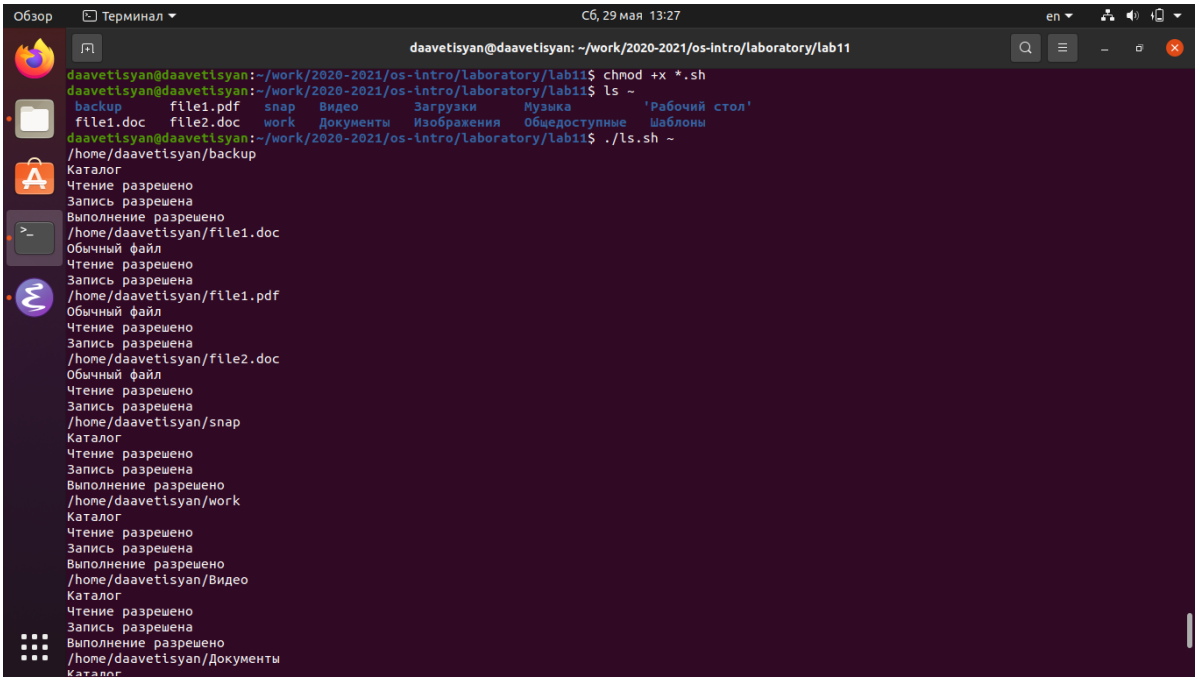

Рис. 3.14: Проверил третий скрипт

4. Для четвертого скрипта также создал файл (команда «touch format.sh») и открыл его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команда «emacs &») (рис. -fig. 3.15).

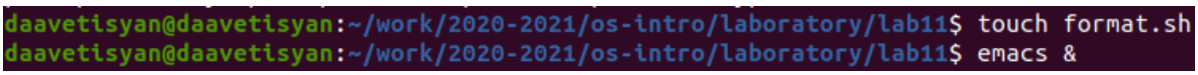

Рис. 3.15: Создал файл для четвёртого скрипта

Написал командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки (рис. -fig. 3.16).

```
#!/bin/bash

b="$1"
shift
for a in $@
do
    k=0
    for i in ${b}/*.${a}
    do
        if test -f "$i"
        then
            let k=k+1
        fi
    done
    echo "$k файлов содержится в каталоге $b с расширением $a"
done
```

Рис. 3.16: Написал четвёртый скрипт

Проверил работу написанного скрипта (команда «./format.sh ~ pdf sh txt doc»), предварительно добавив для него право на выполнение (команда «chmod +x *.sh»), а также создав дополнительные файлы с разными расширениями (команда «touch file.pdf file1.doc file2.doc») (рис. -fig. 3.17). Скрипт работает корректно.

```
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab11$ chmod +x *.sh
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab11$ ls ~
 backup      file1.pdf   snap   Видео       Загрузки      Музыка          'Рабочий стол'
 file1.doc   file2.doc   work   Документы   Изображения   Общедоступные   Шаблоны
daavetisyan@daavetisyan:~/work/2020-2021/os-intro/laboratory/lab11$ ./format.sh ~ pdf doc sh txt
1 файлов содержится в каталоге /home/daavetisyan с расширением pdf
2 файлов содержится в каталоге /home/daavetisyan с расширением doc
0 файлов содержится в каталоге /home/daavetisyan с расширением sh
0 файлов содержится в каталоге /home/daavetisyan с расширением txt
```

Рис. 3.17: Проверил четвёртый скрипт

4 Контрольные вопросы

1. Командный процессор (командная оболочка, интерпретатор команд shell) – это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек:

- оболочка Борна (Bourne shell или sh) – стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;
- C-оболочка (или csh) – надстройка на оболочкой Борна, использующая Сподобный синтаксис команд с возможностью сохранения истории выполнения команд;
- оболочка Корна (или ksh) – напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна;
- BASH – сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation).

2. POSIX (Portable Operating System Interface for Computer Environments) – набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ.
   Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна.

3. Командный процессор bash обеспечивает возможность использования переменных типа строка символов. Имена переменных могут быть выбраны пользователем. Пользователь имеет возможность присвоить переменной значение некоторой строки символов. Например, команда«mark=/usr/andy/bin» присваивает значение строки символов /usr/andy/bin переменной mark типа строка символов.

   Значение, присвоенное некоторой переменной, может быть впоследствии использовано. Для этого в соответствующем месте командной строки должно быть употреблено имя этой переменной, которому предшествует метасимвол $. Например, команда «mv afile ${mark}» переместит файл afile из текущего каталога в каталог с абсолютным полным именем /usr/andy/bin.

   Оболочка bash позволяет работать с массивами. Для создания массива используется команда set с флагом -A. За флагом следует имя переменной, а затем список значений, разделённых пробелами. Например, «set -A states Delaware Michigan “New Jersey”»

   Далее можно сделать добавление в массив, например, states[49]=Alaska. Индексация массивов начинается с нулевого элемента.
4. Оболочка bash поддерживает встроенные арифметические функции. Команда let является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение – это единичный терм (term), обычно целочисленный. Команда let берет два операнда и присваивает их переменной.

   Команда read позволяет читать значения переменных со стандартного ввода:

   «echo “Please enter Month and Day of Birth ?”»

   «read mon day trash»

   В переменные mon и day будут считаны соответствующие значения, введённые с клавиатуры, а переменная trash нужна для того, чтобы отобрать всю избыточно введённую информацию и игнорировать её.

5. В языке программирования bash можно применять такие арифметические операции как сложение (+), вычитание (-), умножение(*), целочисленное деление (/) и целочисленный остаток от деления (%).
6. В (( )) можно записывать условия оболочки bash, а также внутри двойных скобок можно вычислять арифметические выражения и возвращать результат.
7. Стандартные переменные:

- PATH: значением данной переменной является список каталогов, в которых командный процессор осуществляет поиск программы или команды, указанной в командной строке, в том случае, если указанное имя программы или команды не содержит ни одного символа /. Если имя команды содержит хотя бы один символ /, то последовательность поиска, предписываемая значением переменной PATH, нарушается. В этом случае в зависимости от того, является имя команды абсолютным или относительным, поиск начинается соответственно от корневого или текущего каталога.
- PS1 и PS2: эти переменные предназначены для отображения промптера командного процессора. PS1 – это промптер командного процессора, по умолчанию его значение равно символу $ или #. Если какая-то интерактивная программа, запущенная командным процессором, требует ввода, то используется промптер PS2. Он по умолчанию имеет значение символа >.
- HOME: имя домашнего каталога пользователя. Если команда cd вводится без аргументов, то происходит переход в каталог, указанный в этой переменной.
- IFS: последовательность символов, являющихся разделителями в командной строке, например, пробел, табуляция и перевод строки (new line).
- MAIL: командный процессор каждый раз перед выводом на экран промптера проверяет содержимое файла, имя которого указано вэтой переменной, и если содержимое этого файла изменилось с момента последнего ввода из него, то перед тем как вывести на терминал промптер, командный

процессор выводит на терминал сообщение You have mail (у Вас есть почта).

- TERM: тип используемого терминала.
- LOGNAME: содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему.

8. Такие символы, как ’ < > * ? | ” &, являются метасимволами и имеют для командного процессора специальный смысл.
9. Снятие специального смысла с метасимвола называется экранированием метасимвола. Экранирование может быть осуществлено с помощью предшествующего метасимволу символа , который, в свою очередь, является метасимволом.

   Для экранирования группы метасимволов нужно заключить её в одинарные кавычки. Строка, заключённая в двойные кавычки, экранирует все метасимволы, кроме $, ’ , , ”. Например,

   – echo * выведет на экран символ *,*

   – *echo ab’|’cd выведет на экран строку ab*|*cd.

10. Последовательность команд может быть помещена в текстовый файл. Такой файл называется командным. Далее этот файл можно выполнить по команде: «bash командный_файл [аргументы]»

    Чтобы не вводить каждый раз последовательности символов bash, необходимо изменить код защиты этого командного файла, обеспечив доступ к этому файлу по выполнению. Это может быть сделано с помощью команды «chmod +x имя_файла»

    Теперь можно вызывать свой командный файл на выполнение, просто вводя его имя с терминала так, как будто он является выполняемой программой. Командный процессор распознает, что в Вашем файле на самом деле хранится не выполняемая программа, а программа, написанная на языке программирования оболочки, и осуществит еёинтерпретацию.
11. Группу команд можно объединить в функцию. Для этого существует ключе-

вое слово function, после которого следует имя функции и список команд, заключённых в фигурные скобки. Удалить функцию можно с помощью команды unset с флагом -f.

12. Чтобы выяснить, является ли файл каталогом или обычным файлом, необходимо воспользоваться командами «test -f [путь до файла]» (для проверки, является ли обычным файлом) и «test -d [путь до файла]» (для проверки, является ли каталогом).
13. Команду «set» можно использовать для вывода списка переменных окружения. В системах Ubuntu и Debian команда «set» также выведет список функций командной оболочки после списка переменных командной оболочки. Поэтому для ознакомления со всеми элементами списка переменных окружения при работе с данными системами рекомендуется использовать команду «set | more».

    Команда «typeset» предназначена для наложения ограничений на переменные.

    Команду «unset» следует использовать для удаления переменной из окружения командной оболочки.
14. При вызове командного файла на выполнение параметры ему могут быть переданы точно таким же образом, как и выполняемой программе. С точки зрения командного файла эти параметры являются позиционными. Символ $ является метасиволом процессора. Он используется, в частности, для ссылки на параметры, точнее, для получения их значений в командном файле. В командный файл можно передать до девяти параметров. При использовании где-либо в командном файле комбинации символов $i, где 0 < i < 10, вместо неё будет осуществлена подстановка значения параметра с порядковым номером i, т. е. аргумента командного файла с порядковым номером i. Использование комбинации символов $0 приводит к подстановкевместо неё имени данного командного файла.
15. Специальные переменные:

- $* – отображается вся командная строка или параметры оболочки;
- $? – код завершения последней выполненной команды;
- $$ – уникальный идентификатор процесса, в рамках которого выполняется командный процессор;
- $! – номер процесса, в рамках которого выполняется последняя вызванная на выполнение в командном режиме команда;
- $- – значение флагов командного процессора;
- ${*#} – возвращает целое число – количество слов, которые были результатом $*;
- ${#name} – возвращает целое значение длины строки в переменной name;
- ${name[n]} – обращение к n-му элементу массива;
- ${name[*]} – перечисляет все элементы массива, разделённые пробелом;
- ${name[@]} – то же самое, но позволяет учитывать символы пробелы в самих переменных;
- ${name:-value} – если значение переменной name не определено, то оно будет заменено на указанное value;
- ${name:value} – проверяется факт существования переменной;
- ${name=value} – если name не определено, то ему присваивается значение value;
- ${name?value} – останавливает выполнение, если имя переменной не определено, и выводит value как сообщение об ошибке;
- ${name+value} – это выражение работает противоположно ${name-value}. Если переменная определена, то подставляется value;
- ${name#pattern} – представляет значение переменной name с удалённым самым коротким левым образцом (pattern);
- ${#name[*]} и ${#name[@]} – эти выражения возвращают количество элементов в массиве name.

5 Выводы

В ходе выполнения данной лабораторной работы я изучил основы программирования в оболочке ОС UNIX/Linux и научился писать небольшие командные файлы.