

Лабораторная работа №10

Дисциплина: Операционные системы

Кабанова Варвара

Содержание

| | |
|---|-----------|
| 1 Цель работы | 5 |
| 2 Выполнение лабораторной работы | 6 |
| 3 Ответы на контрольные вопросы | 15 |
| 4 Выводы | 21 |

List of Figures

| | | |
|------|--------|----|
| 2.1 | рис.1 | 6 |
| 2.2 | рис.2 | 6 |
| 2.3 | рис.3 | 7 |
| 2.4 | рис.4 | 7 |
| 2.5 | рис.5 | 8 |
| 2.6 | рис.6 | 9 |
| 2.7 | рис.7 | 9 |
| 2.8 | рис.8 | 10 |
| 2.9 | рис.9 | 10 |
| 2.10 | рис.10 | 11 |
| 2.11 | рис.11 | 11 |
| 2.12 | рис.12 | 12 |
| 2.13 | рис.13 | 12 |
| 2.14 | рис.14 | 13 |
| 2.15 | рис.15 | 13 |
| 2.16 | рис.16 | 14 |
| 2.17 | рис.17 | 14 |

List of Tables

1 Цель работы

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

2 Выполнение лабораторной работы

Для начала я изучила команды архивации, используя команды «man zip», «man bzip2», «man tar» (рис.1-4).

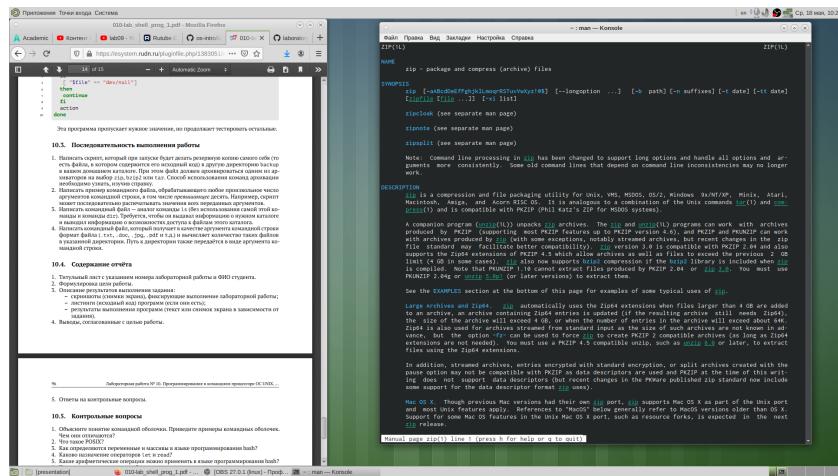


Figure 2.1: рис.1

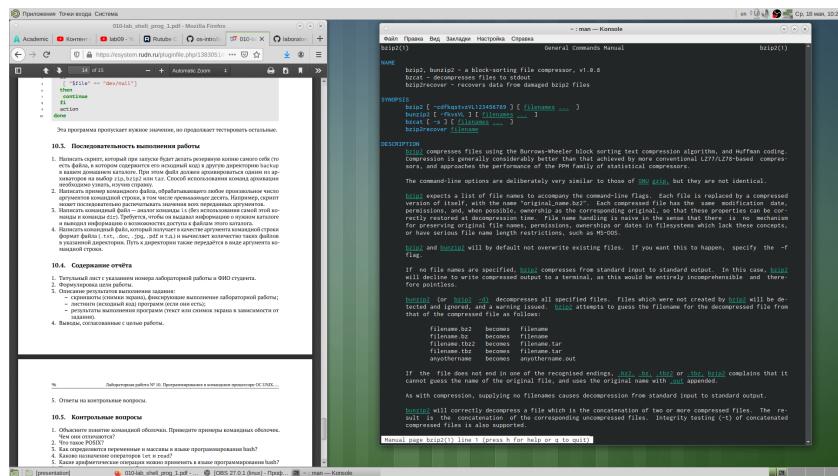


Figure 2.2: рис.2

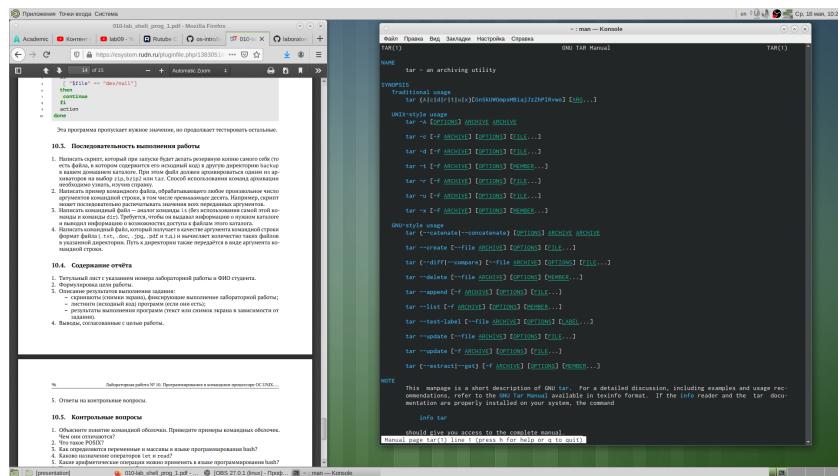


Figure 2.3: рис.3

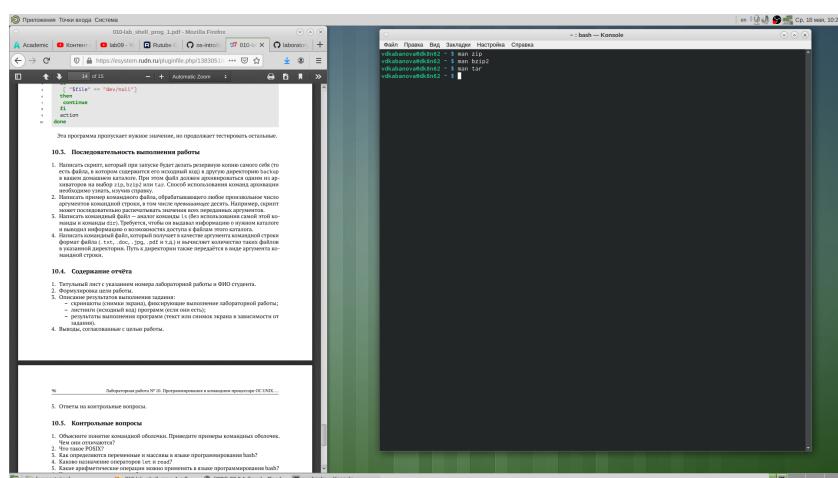


Figure 2.4: рис.4

- Синтаксис команды zip для архивации файла: `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]`
- Синтаксис команды bzip2 для архивации файла: `bzip2 [опции] [имена файлов]`. Синтаксис команды bzip2 для разархивации/распаковки файла: `bunzip2[опции] [архивы.bz2]`
- Синтаксис команды tar для архивации файла: `tar[опции][архив.tar][файлы]`

лы_для_архивации]. Синтаксис команды tar для разархивации/распаковки файла: tar[опции][архив.tar]

Создала файл, в котором буду писать первый скрипт, и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch backup.sh» и «emacs &») (рис.5-6).

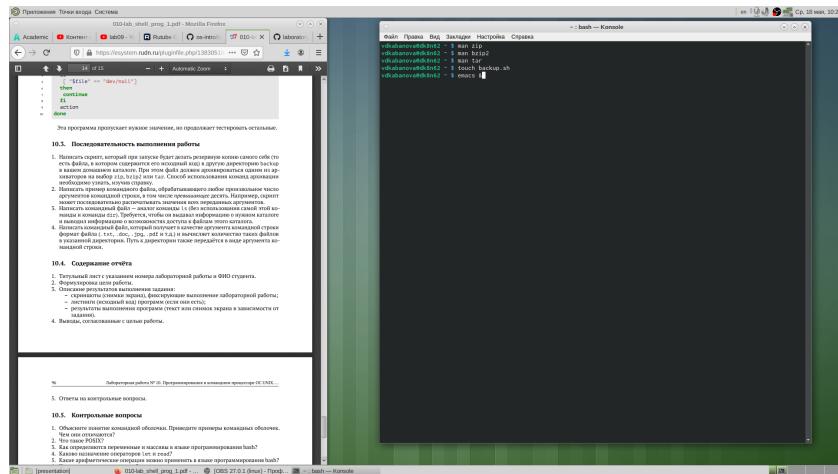


Figure 2.5: рис.5

Написала скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию back up в вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов zip, bzip2 или tar (рис.6). При написании скрипта использовала архиватор bzip2.

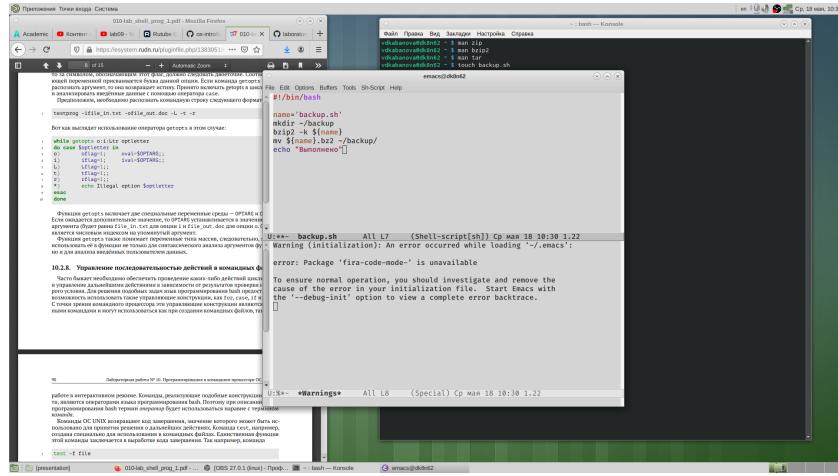


Figure 2.6: рис.6

Проверила работу скрипта (команда «./backup.sh»), предварительно добавив для него право на выполнение (команда «chmod+x*.sh»). Проверила, появился ли каталог backup/, перейдя в него (команда «cd backup/»), посмотрела его содержимое (команда «ls») и просмотрела содержимое архива (команда «bunzip2 -c backup.sh.bz2») (рис.7). Скрипт работает корректно.

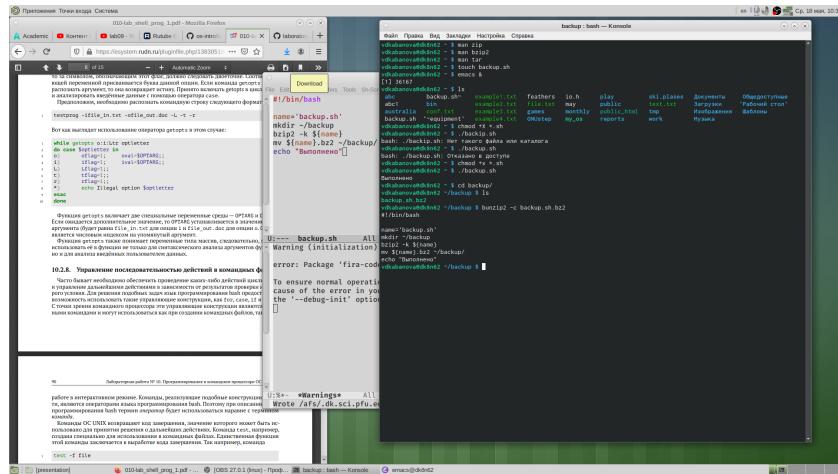


Figure 2.7: рис.7

Создала файл, в котором буду писать второй скрипт, и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch prog2.sh» и «emacs &>») (рис.8).

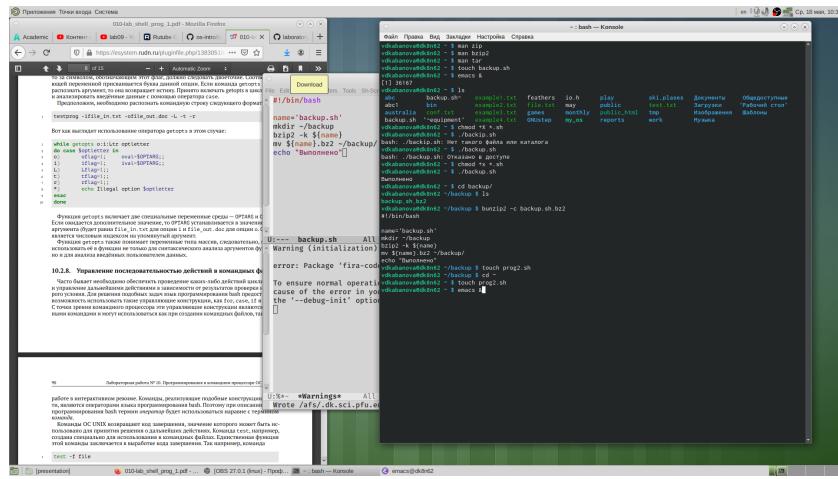


Figure 2.8: рис.8

Написала 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.9).

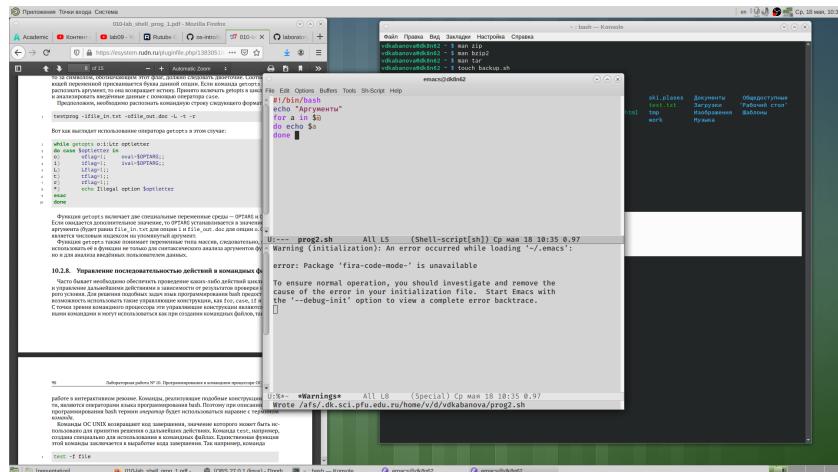


Figure 2.9: рис.9

Проверила работу написанного скрипта (команды «./prog2.sh 1 2 3 4» и «./prog2.sh 1 2 3 4 5 6 7 8 9 10 11»), предварительно добавив для него право на выполнение (команда «chmod+x*.sh»). Вводила аргументы количество которых меньше 10 и больше 10 (рис.10). Скрипт работает корректно.

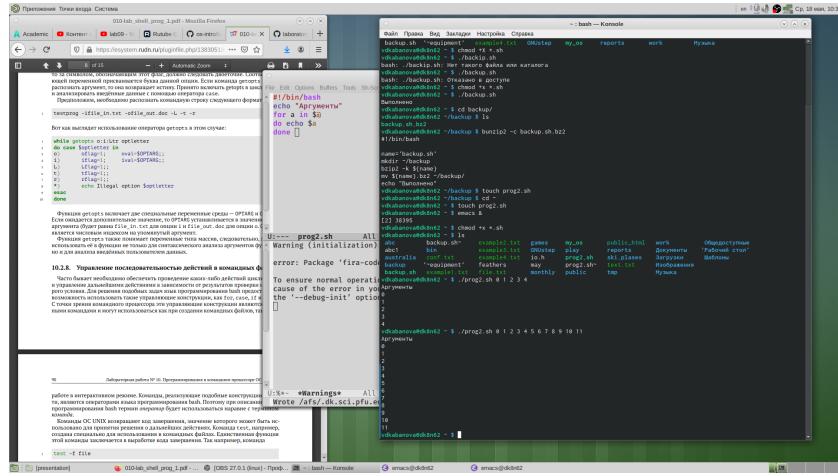


Figure 2.10: рис.10

Создала файл, в котором буду писать третий скрипт, и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touchprogl.sh» и «emacs&») (рис.11).

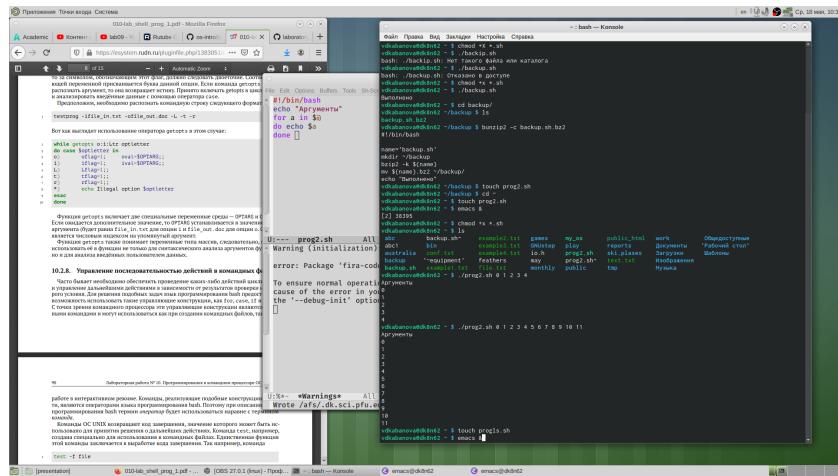


Figure 2.11: рис.11

Написала командный файл – аналог команды ls (без использования самой этой команды и команды dir). Он должен выдавать информацию о нужном каталоге и выводить информацию о возможностях доступа к файлам этого каталога (рис.12).

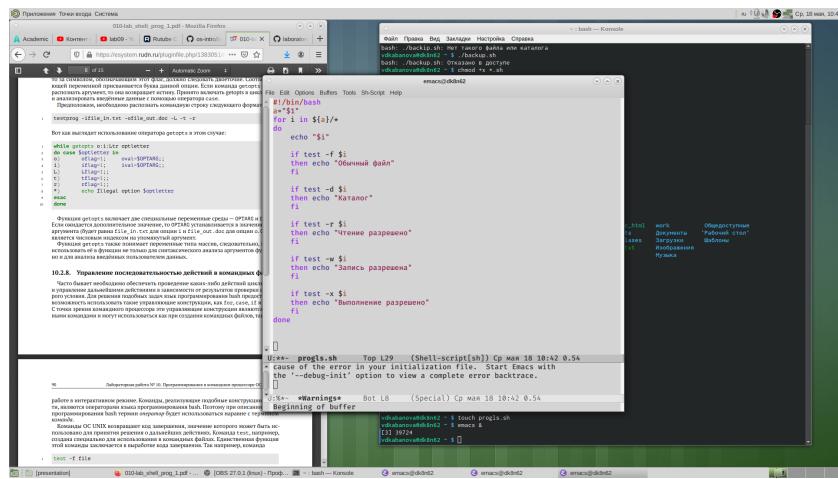


Figure 2.12: рис.12

Далее проверила работу скрипта (команда «./progls.sh»), предварительно добавив для него право на выполнение (команда «chmod+x*.sh») (рис.13-14). Скрипт работает корректно.

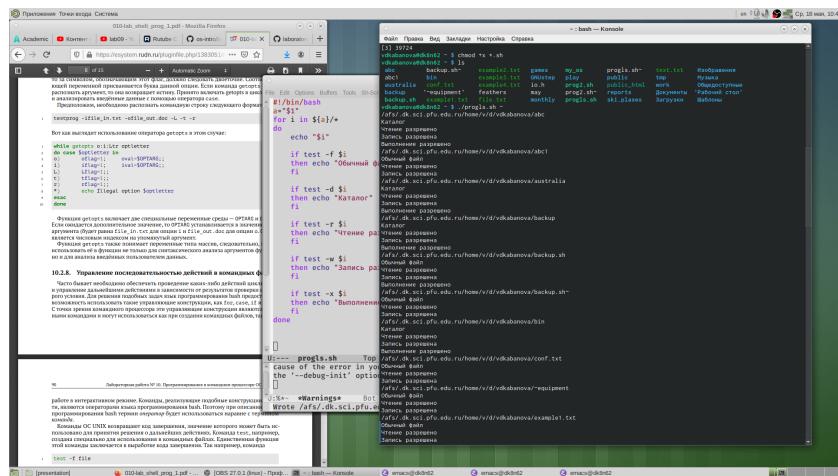


Figure 2.13: рис.13

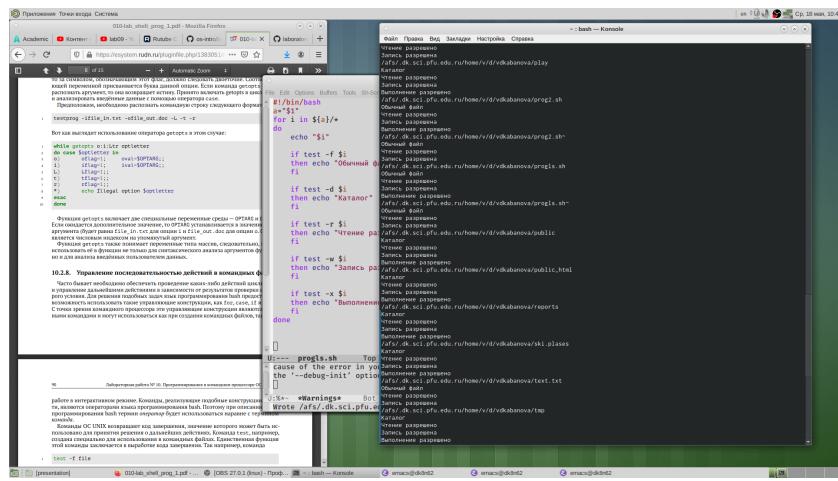


Figure 2.14: рис.14

Для четвертого скрипта создала файл (команда «touch format.sh») и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команда «emacs &») (рис.15).

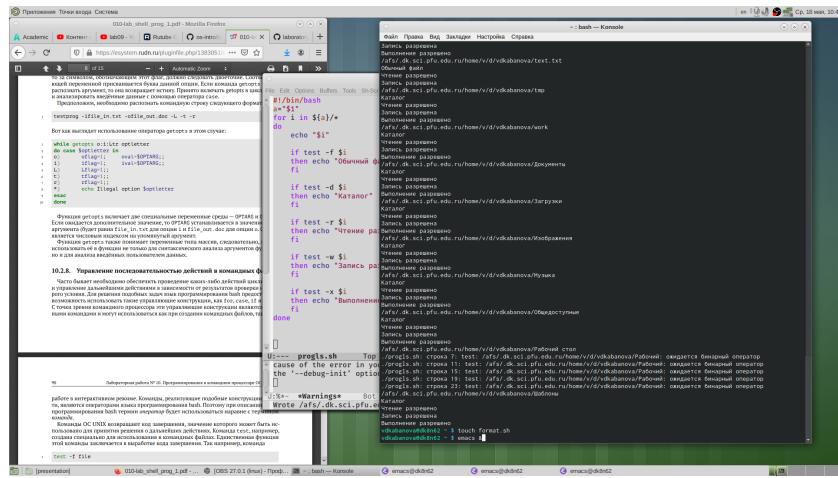


Figure 2.15: рис.15

Написала 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.16).

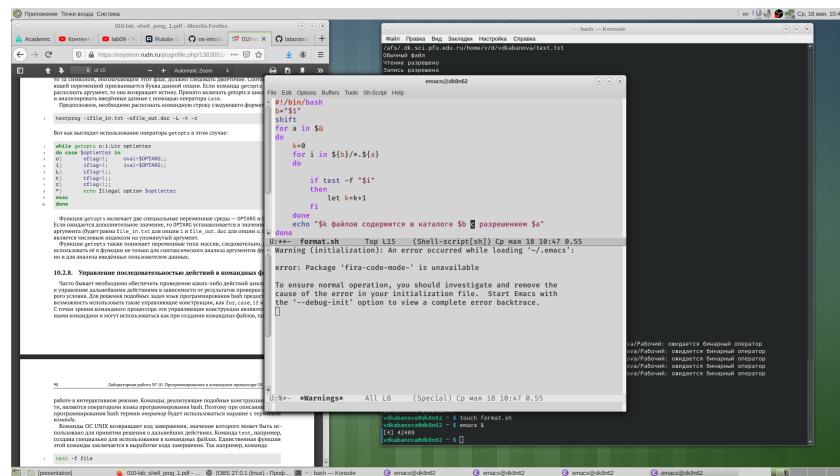


Figure 2.16: рис.16

Проверила 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.17). Скрипт работает корректно.

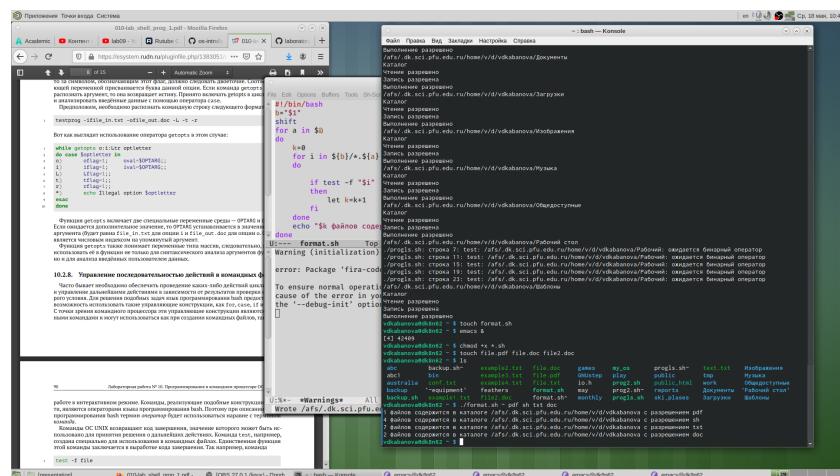


Figure 2.17: рис.17

3 Ответы на 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:
 1. оболочка Борна (BourneShell или sh) – стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;
 2. С-оболочка (или csh) – надстройка на оболочкой Борна, использующая Сподобный синтаксис команд с возможностью сохранения истории выполнения команд;
 3. Оболочка Корна (или ksh) – напоминает оболочку С, но операторы управления программой совместимы с операторами оболочки Борна;
 4. BASH – сокращение от BourneAgainShell(опять оболочка Борна), в основе своей совмещает свойства оболочек С и Корна (разработка компании FreeSoftwareFoundation).
- 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 `.`, «`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` 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` `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:
 1. 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 корневогоили текущего каталога.
 2. 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 >.
 3. HOME: имя домашнего каталога пользователя. Если команда cdводится без аргументов, то происходит переход в каталог,указанный в этой переменной.
 4. IFS:последовательность символов, являющихся разделителями в командной строке, например, пробел, табуляция и перевод строки (newline).
 5. MAIL:командный процессор каждый раз перед выводом на экран промптера проверяет содержимое файла, имя которого указано в этой 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 havemail(у Вас есть почта).

6. TERM: тип используемого терминала.
7. 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 мета символа. Экранирование может быть осуществлено с помощью предшествующего мета символу символа , который, в свою очередь, является мета символом. Для экранирования группы метасимволов нужно заключить её в одинарные кавычки. Стока, 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 осуществить её интерпретацию.
- 11) Группу команд можно объединить в функцию. Для этого существует ключевое 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 `$` является метасимволом командного процессора. Он используется, в частности, для ссылки на параметры, точнее, для получения их значений в командном файле. В командный файл можно передать до девяти параметров. При использовании где-либо в командном файле комбинации символов `$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 `$0` приводит к подстановке вместо неё имени данного командного файла.
- 15) Специальные переменные:
 1. `$*` – отображается вся командная строка или параметры оболочки;
 2. `$?` – код завершения последней выполненной команды;
 3. `$$` – уникальный идентификатор процесса, в рамках которого выполняется командный процессор;

4. $\$!$ –номер процесса, в рамках которого выполняется последняя вызванная на выполнение в командном режиме команда;
5. $\$-$ значение флагов командного процессора;
6. $\$\{#\}$ –возвращает целое число –количество слов, которые были результатом $\$$;
7. $\$\{#name\}$ –возвращает целое значение длины строки в переменной name;
8. $\$\{name[n]\}$ –обращение к n-му элементу массива;
9. $\$\{name[*]\}$ –перечисляет все элементы массива, разделённые пробелом;
10. $\$\{name[@]\}$ –то же самое, но позволяет учитывать символы пробелы в самих переменных;
11. $\$\{name:-value\}$ –если значение переменной name не определено, то оно будет заменено на указанное value;
12. $\$\{name:value\}$ –проверяется факт существования переменной;
13. $\$\{name=value\}$ –если name не определено, то ему присваивается значение value;
14. $\$\{name?value\}$ –останавливает выполнение, если имя переменной не определено, и выводит value как сообщение об ошибке;
15. $\$\{name+value\}$ –это выражение работает противоположно $\$\{name-value\}$. Если переменная определена, то подставляется value;
16. $\$\{name#pattern\}$ –представляет значение переменной name с удалённым самым коротким левым образцом (pattern);
17. $\$\{#name[*]\}$ и $\$\{#name[@]\}$ –эти выражения возвращают количество элементов в массиве name.

4 Выводы

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