

Отчет по лабораторной работе №10

Операционные системы

Морозова Ульяна Константиновна

Список иллюстраций

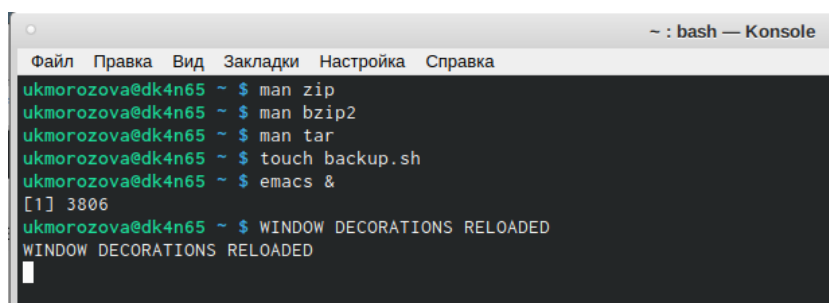
0.1. Команда man	4
0.2. Архив zip	5
0.3. Архив bzip2	5
0.4. Архив tar	6
0.5. Создание файла	6
0.6. Программа	7
0.7. Права доступа	7
0.8. Просмотр каталога	7
0.9. Просмотр содержимого	8
0.10. Создание файла	8
0.11. Программа	9
0.12. Выполнение	9
0.13. Выполнение	9
0.14. Код программы	10
0.15. Запуск файла	11
0.16. Создание файла	11
0.17. Код программы	12
0.18. Запуск файла	12

Цель работы

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

Выполнение лабораторной работы

1. Для начала изучим команды архивации с помощью команд `man bzip2`, `man zip`, `man tar` (рис.1).



```
ukmorofova@dk4n65 ~ $ man zip
ukmorofova@dk4n65 ~ $ man bzip2
ukmorofova@dk4n65 ~ $ man tar
ukmorofova@dk4n65 ~ $ touch backup.sh
ukmorofova@dk4n65 ~ $ emacs &
[1] 3806
ukmorofova@dk4n65 ~ $ WINDOW DECORATIONS RELOADED
WINDOW DECORATIONS RELOADED
```

Рис. 0.1.: Команда `man`

Синтаксис команды `zip` для архивации файлов: `zip [опции] [имя_файла.zip]` [файлы или папки для архивации]. Синтаксис для разархивации файла: `unzip [опции] [файл_архива.zip] [файлы] -x [исключить] -d [папки]` (рис.2).

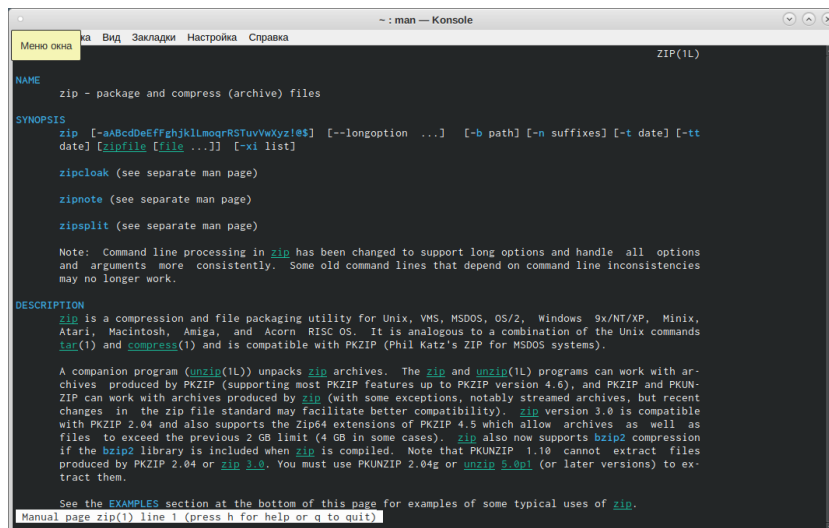


Рис. 0.2.: Архив zip

Синтаксис команды bzip2 для архивации файлов: `bzip2 [опции] [имя_файла]`.
 Синтаксис для разархивации файла: `bunzip2 [опции] [файл_архива.bz2]` (рис.3).

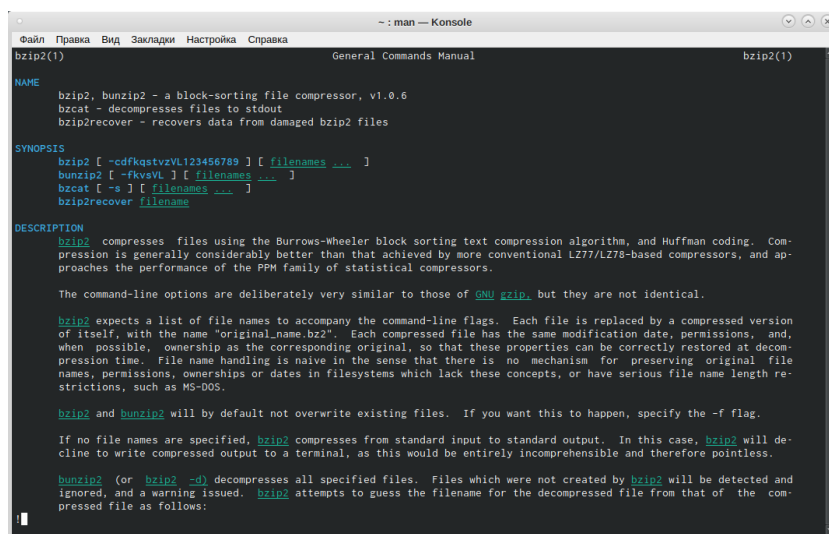


Рис. 0.3.: Архив bzip2

Синтаксис команды tar для архивации файлов: `tar [опции] [архив.tar]`. Синтаксис для разархивации файла: `tar [опции] [архив.tar]` (рис.4).

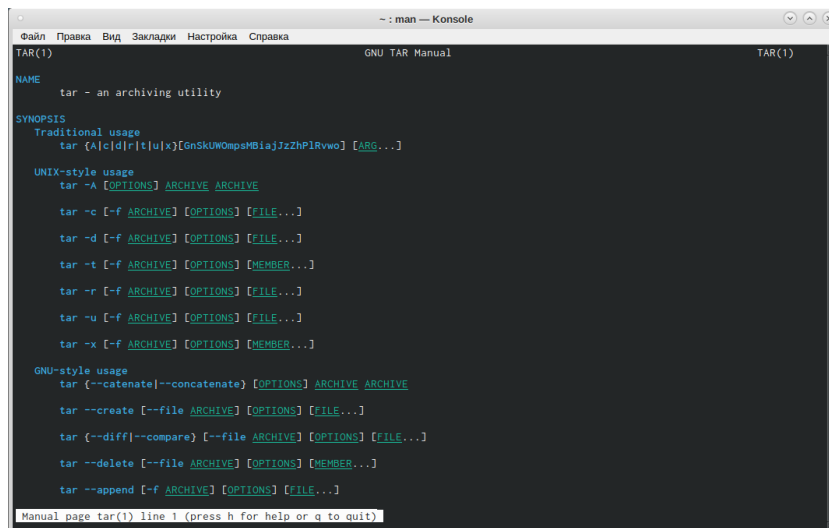


Рис. 0.4.: Архив tar

Создаем файл backup.sh, в котором напишем первый скрипт, и откроем в редакторе emacs (рис.5)

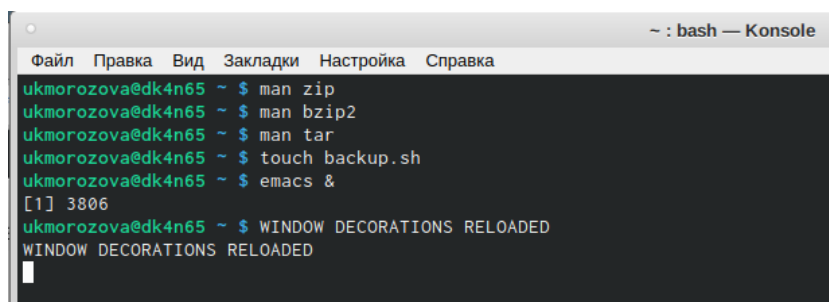
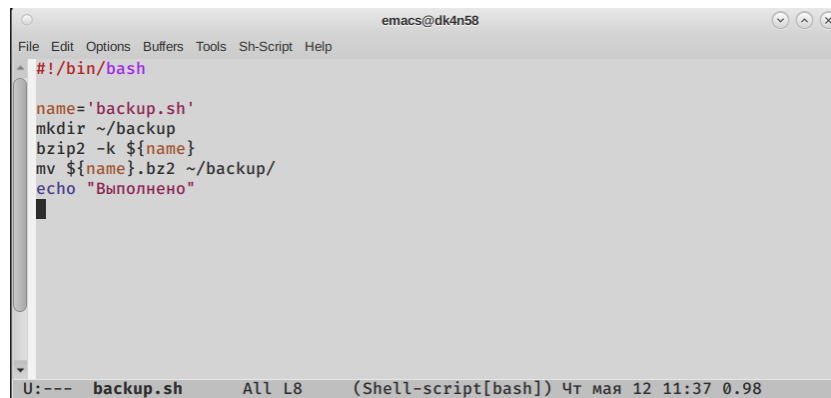


Рис. 0.5.: Создание файла

Напишем программу, которая при запуске будет делать резервную копию самого себя в другую директорию backup в нашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar (выбираем bzip2) (рис.6).



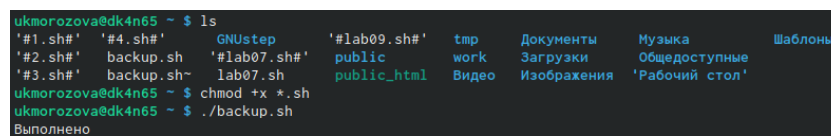
```
emacs@dk4n58
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash

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

U:--- backup.sh All L8 (Shell-script[bash]) Чт мая 12 11:37 0.98

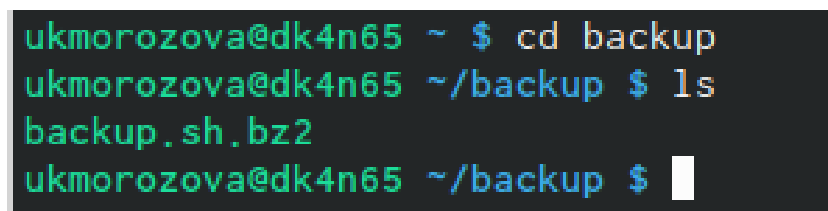
Рис. 0.6.: Программа

Проверим работу скрипта (команда `./backup.sh`), предварительно добавив для него права на выполнение (`chmod +x *.sh`) (рис.7). Перейдем в каталог `backup` и удостоверимся, что файл появился в этом каталоге, (рис.8) и просмотрим содержимое архива (команда `bunzip2 - backup.sh.bz2`) (рис.9).



```
ukmorofova@dk4n65 ~ $ ls
'#1.sh#' '#4.sh#' GNUstep '#lab09.sh#' tmp Документы Музыка Шаблоны
'#2.sh#' backup.sh '#lab07.sh#' public work Загрузки Общедоступные
'#3.sh#' backup.sh lab07.sh public_html Видео Изображения 'Рабочий стол'
ukmorofova@dk4n65 ~ $ chmod +x *.sh
ukmorofova@dk4n65 ~ $ ./backup.sh
Выполнено
```

Рис. 0.7.: Права доступа



```
ukmorofova@dk4n65 ~ $ cd backup
ukmorofova@dk4n65 ~/backup $ ls
backup.sh.bz2
ukmorofova@dk4n65 ~/backup $
```

Рис. 0.8.: Просмотр каталога

```

ukmorozova@dk4n65 ~/backup $ bunzip2 -c backup.sh.bz2
#!/bin/bash

name='backup.sh'
mkdir ~/backup
bzip2 -k ${name}
mv ${name}.bz2 ~/backup/
echo "Выполнено"
ukmorozova@dk4n65 ~/backup $

```

Рис. 0.9.: Просмотр содержимого

2. Создадим файл `prog.sh`, в котором напомним второй скрипт, и откроем его в `emacs` (рис.10).

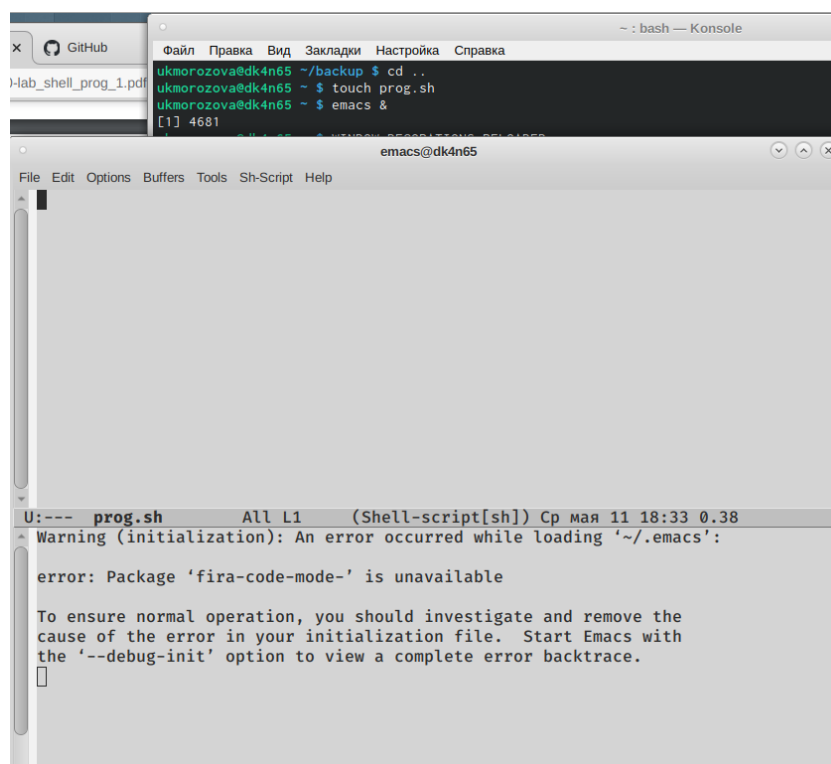



Рис. 0.10.: Создание файла

Напишем 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.11).



```
emacs@dk4n65
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash
echo "Аргументы"
for a in $@
do echo $a
done
U:--- prog.sh All L1 (Shell-script[bash]) Cp ma
```

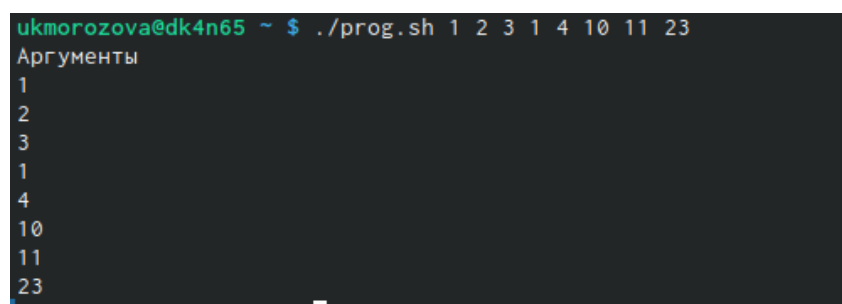
Рис. 0.11.: Программа

Проверим его работу, предварительно передав права доступа на выполнение, вводим разное количество аргументов меньше и больше 10 (рис.12-13).



```
ukmorozova@dk4n65 ~ $ chmod +x *.sh
ukmorozova@dk4n65 ~ $ ls
'1.sh#' '4.sh#' backup.sh~ lab07.sh prog.sh~ tmp Документы Музыка Шаблоны
'2.sh#' backup GNUstep '#lab09.sh#' public work Загрузки Общедоступные
'3.sh#' backup.sh '#lab07.sh#' prog.sh public_html Видео Изображения 'Рабочий стол'
ukmorozova@dk4n65 ~ $ ./prog.sh 1 2 3 1 4
Аргументы
1
2
3
1
4
```

Рис. 0.12.: Выполнение



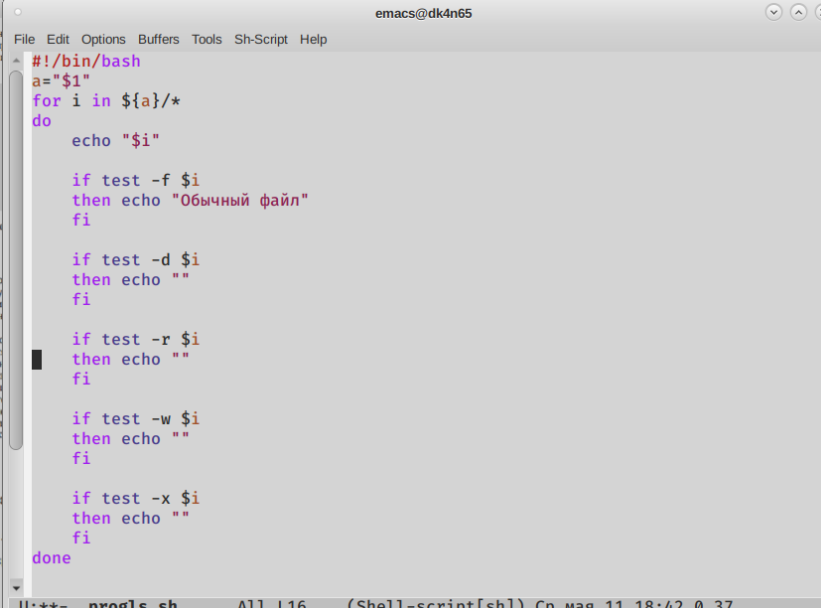
```
ukmorozova@dk4n65 ~ $ ./prog.sh 1 2 3 1 4 10 11 23
Аргументы
1
2
3
1
4
10
11
23
```

Рис. 0.13.: Выполнение

3. Создадим файл progl.sh для третьего скрипта (команда touch progl.sh) и

откроем его в emacs.

Напишем командный файл — аналог команды ls (без использования самой этой команды и команды dir) (рис.14). Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога.



```
emacs@dk4n65
File Edit Options Buffers Tools Sh-Script Help
^
#!/bin/bash
a="$1"
for i in ${a}/*
do
    echo "$i"

    if test -f $i
    then echo "Обычный файл"
    fi

    if test -d $i
    then echo ""
    fi

    if test -r $i
    then echo ""
    fi

    if test -w $i
    then echo ""
    fi

    if test -x $i
    then echo ""
    fi
done
U:***- proglis.sh    All L16    (Shell-script[sh]) Ср мая 11 18:42 0.37
```

Рис. 0.14.: Код программы

Проверим его работу, предварительно передав права доступа на выполнение (chmod +x *.sh) и командой ./proglis.sh запустим его (рис.15).

```
~ : bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
10
11
23
ukmorozova@dk4n65 ~ $ touch progl.sh
ukmorozova@dk4n65 ~ $ emacs &
[1] 5050
ukmorozova@dk4n65 ~ $ WINDOW DECORATIONS RELOADED
WINDOW DECORATIONS RELOADED

[1]+  Завершён      emacs
ukmorozova@dk4n65 ~ $ chmod +x *.sh
ukmorozova@dk4n65 ~ $ ls
"#1.sh#"  "#4.sh#"  backup.sh~  lab07.sh  progl.sh~  public      work      Загрузки      Общедоступные
"#2.sh#"  backup    GNUstep    '#lab09.sh#'  prog.sh  public_html  Видео      Изображения  'Рабочий стол'
"#3.sh#"  backup.sh  '#lab07.sh#'  progl.sh  prog.sh~  tmp         Документы  Музыка      Шаблоны
ukmorozova@dk4n65 ~ $ ./progl.sh ~
/afs/.dk.sci.pfu.edu.ru/home/u/k/ukmorozova/#1.sh#
Обычный файл

/afs/.dk.sci.pfu.edu.ru/home/u/k/ukmorozova/#2.sh#
Обычный файл

/afs/.dk.sci.pfu.edu.ru/home/u/k/ukmorozova/#3.sh#
Обычный файл

/afs/.dk.sci.pfu.edu.ru/home/u/k/ukmorozova/#4.sh#
Обычный файл

/afs/.dk.sci.pfu.edu.ru/home/u/k/ukmorozova/backup
Обычный файл

/afs/.dk.sci.pfu.edu.ru/home/u/k/ukmorozova/backup.sh
Обычный файл

/afs/.dk.sci.pfu.edu.ru/home/u/k/ukmorozova/backup.sh~
Обычный файл

/afs/.dk.sci.pfu.edu.ru/home/u/k/ukmorozova/GNUstep
Обычный файл

/afs/.dk.sci.pfu.edu.ru/home/u/k/ukmorozova/#lab07.sh#
Обычный файл

/afs/.dk.sci.pfu.edu.ru/home/u/k/ukmorozova/lab07.sh
Обычный файл
```

Рис. 0.15.: Запуск файла

4. Создадим файл `format.sh` для четвертого скрипта (команда `touch format.sh`) и откроем его в `emacs` (рис.16).

```
~ : bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
ukmorozova@dk4n65 ~ $ touch format.sh
ukmorozova@dk4n65 ~ $ emacs &
[1] 5472
ukmorozova@dk4n65 ~ $ WINDOW DECORATIONS RELOADED
WINDOW DECORATIONS RELOADED
```

Рис. 0.16.: Создание файла

Напишем 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.17).

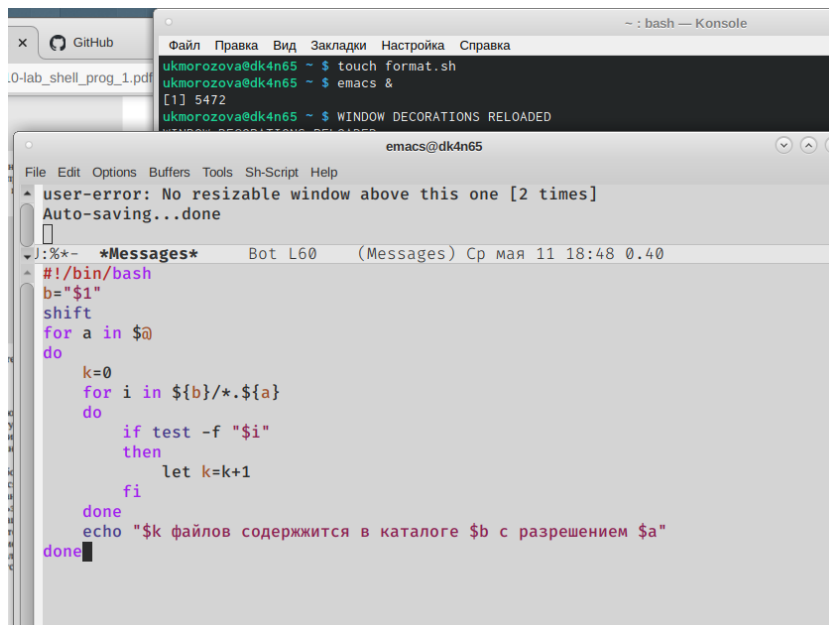


Рис. 0.17.: Код программы

Проверим его работу, предварительно передав права доступа на выполнение (chmod +x *.sh) и командой ./format.sh ~ pdf txt docx запустим его (рис.18).

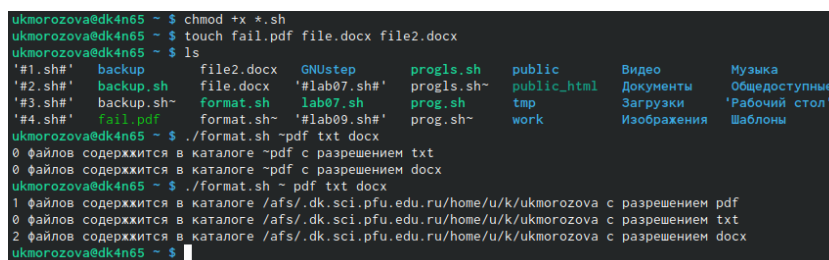


Рис. 0.18.: Запуск файла

Выводы

Я научилась писать небольшие команды в оболочке Linux.

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 оболочке Борна, использующая подобный 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 зна-

чение 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 `.`, «`mva file{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, введенные с клавиатуры, а переменная `trash` нужна для того, чтобы отобрать всю избыточно введенную 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, указан-

ной 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 (newline).

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 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 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 одинар-

ные кавычки. Строка, заключённая в двойные кавычки, экранирует все метасимволы, кроме \$, ', , ". Например, `-echo*` выведет на экран символ , `-echoab'|'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:

\$* –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.