

Отчёт по лабораторной работе №10

**Программирование в командном процессоре ОС UNIX. Командные
файлы**

Федюшина Ярослава Андреевна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	14
5	Контрольные вопросы	15

Список иллюстраций

3.1	man tar	7
3.2	справка пр tar	8
3.3	создание файла и его исполнение	9
3.4	заходим в текстовый редактор	10
3.5	пишем код и сохраняем	11
3.6	скрипт файл	12
3.7	backur	12
3.8	файл	13

Список таблиц

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. Сначала изучаю tar через команду man. Затем пишу скрипт, который при запуске будет делать резервную копию самого себя в другую директорию. При этом файл должен архивироваться одним из архиваторов на выбор zip,bzip2 или tar.

3.1

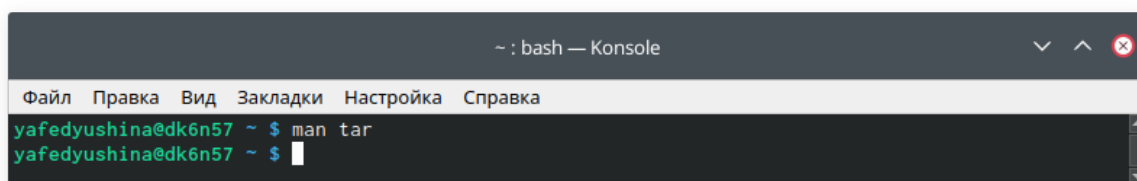
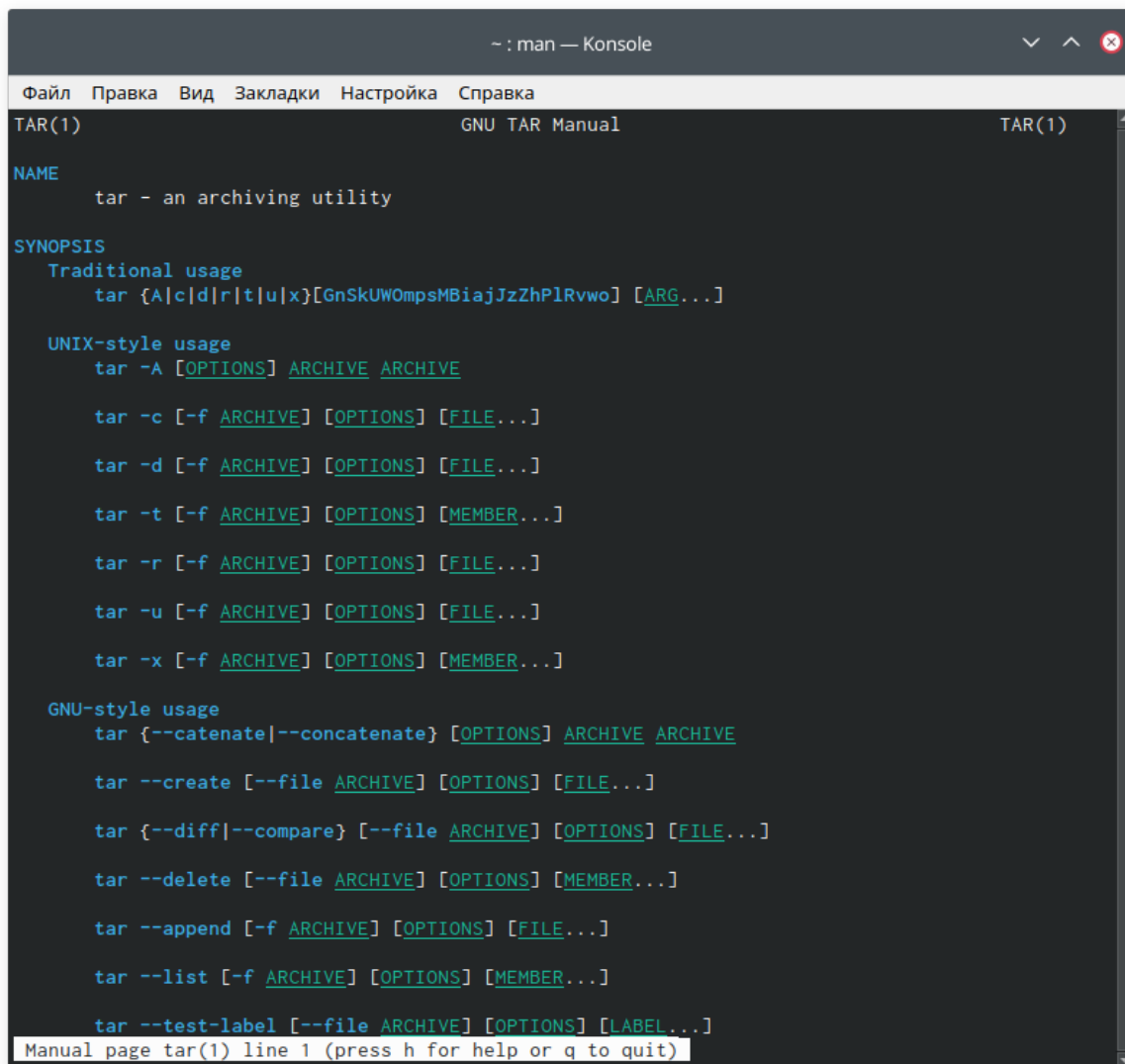


Рис. 3.1: man tar

3.2



The image shows a terminal window titled '~ : man — Konsole'. The window displays the manual page for the 'tar' command. The menu bar at the top includes 'Файл', 'Правка', 'Вид', 'Закладки', 'Настройка', and 'Справка'. The title bar shows 'TAR(1)' on the left and right, and 'GNU TAR Manual' in the center. The content of the manual page is as follows:

```
NAME
    tar - an archiving utility

SYNOPSIS
    Traditional usage
        tar {A|c|d|r|t|u|x}[GnSkUWompsMBiajJzZhPlRvwo] [ARG...]

    UNIX-style usage
        tar -A [OPTIONS] ARCHIVE ARCHIVE

        tar -c [-f ARCHIVE] [OPTIONS] [FILE...]
        tar -d [-f ARCHIVE] [OPTIONS] [FILE...]
        tar -t [-f ARCHIVE] [OPTIONS] [MEMBER...]
        tar -r [-f ARCHIVE] [OPTIONS] [FILE...]
        tar -u [-f ARCHIVE] [OPTIONS] [FILE...]
        tar -x [-f ARCHIVE] [OPTIONS] [MEMBER...]

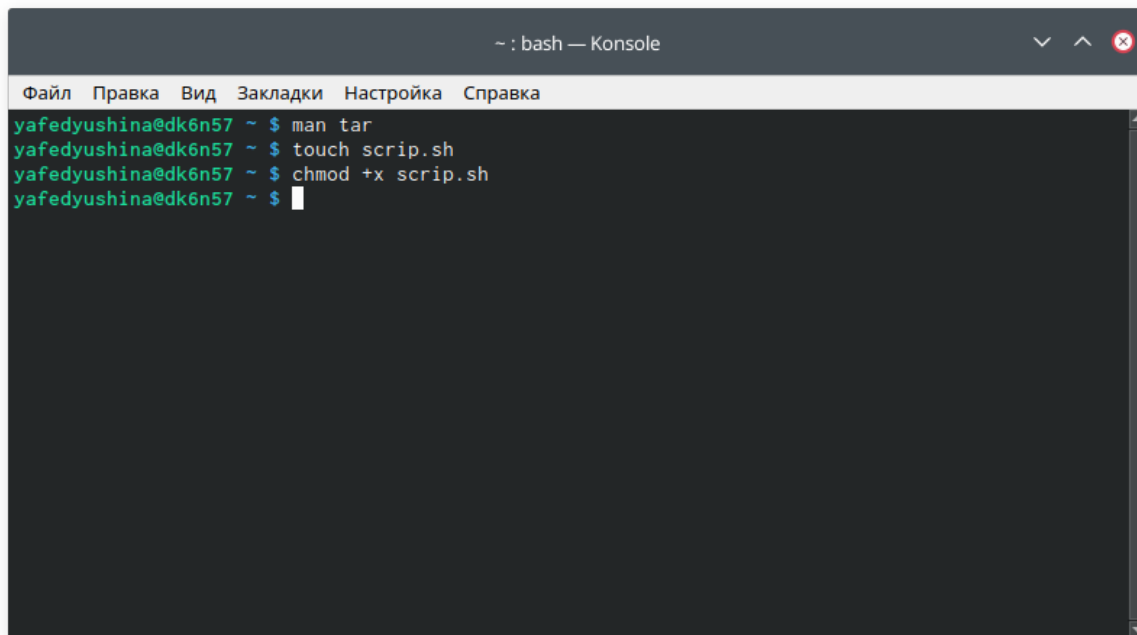
    GNU-style usage
        tar {--catenate|--concatenate} [OPTIONS] ARCHIVE ARCHIVE

        tar --create [--file ARCHIVE] [OPTIONS] [FILE...]
        tar {--diff|--compare} [--file ARCHIVE] [OPTIONS] [FILE...]
        tar --delete [--file ARCHIVE] [OPTIONS] [MEMBER...]
        tar --append [-f ARCHIVE] [OPTIONS] [FILE...]
        tar --list [-f ARCHIVE] [OPTIONS] [MEMBER...]
        tar --test-label [--file ARCHIVE] [OPTIONS] [LABEL...]

Manual page tar(1) line 1 (press h for help or q to quit)
```

Рис. 3.2: справка пр tar

3.3



The image shows a terminal window titled '~ : bash — Konsole'. The window has a menu bar with options: 'Файл', 'Правка', 'Вид', 'Закладки', 'Настройка', and 'Справка'. The terminal content shows a user named 'yafedyushina@dk6n57' performing the following commands:

```
yafedyushina@dk6n57 ~ $ man tar
yafedyushina@dk6n57 ~ $ touch scrip.sh
yafedyushina@dk6n57 ~ $ chmod +x scrip.sh
yafedyushina@dk6n57 ~ $
```

Рис. 3.3: создание файла и его исполнение

3.4

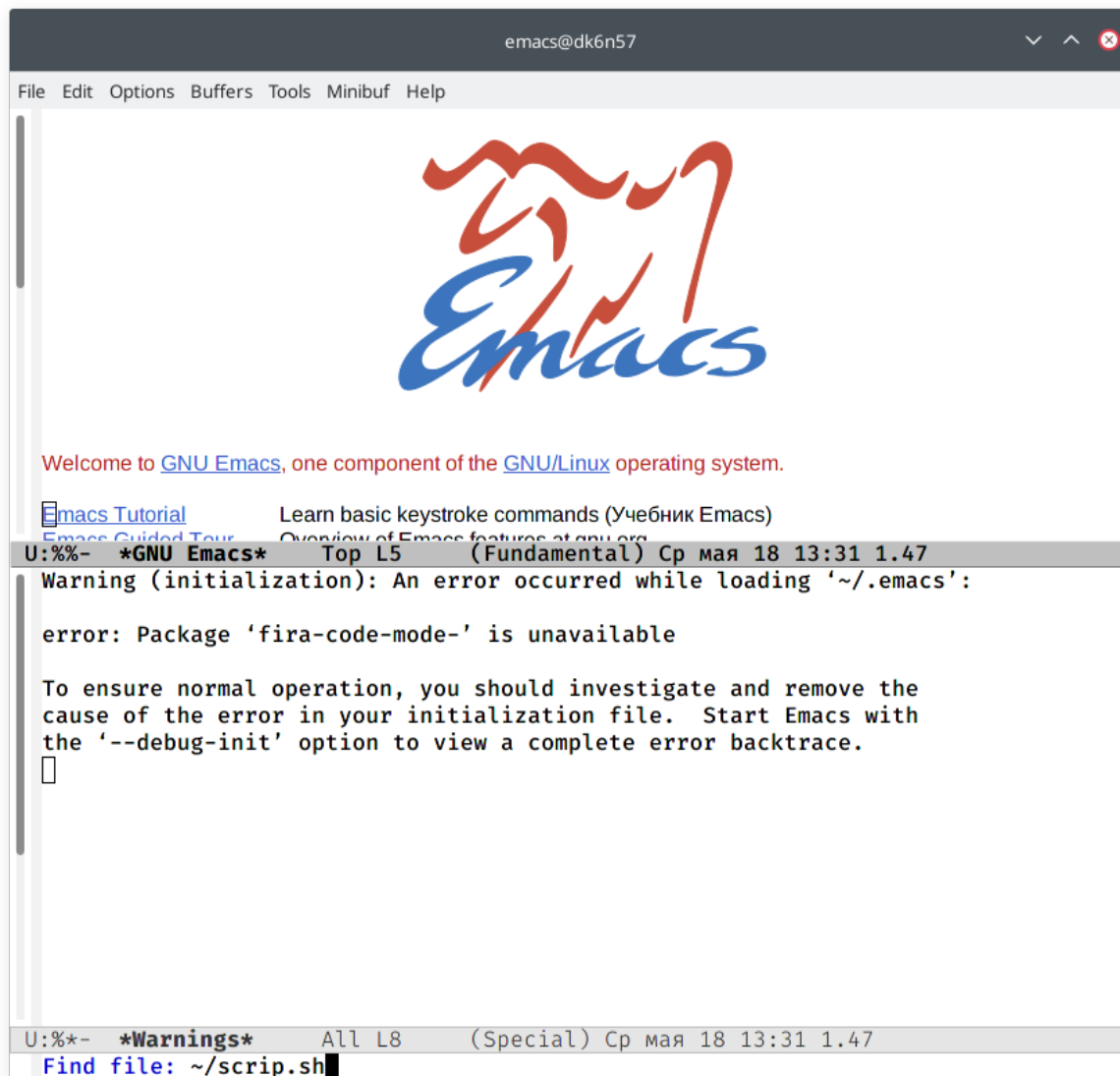
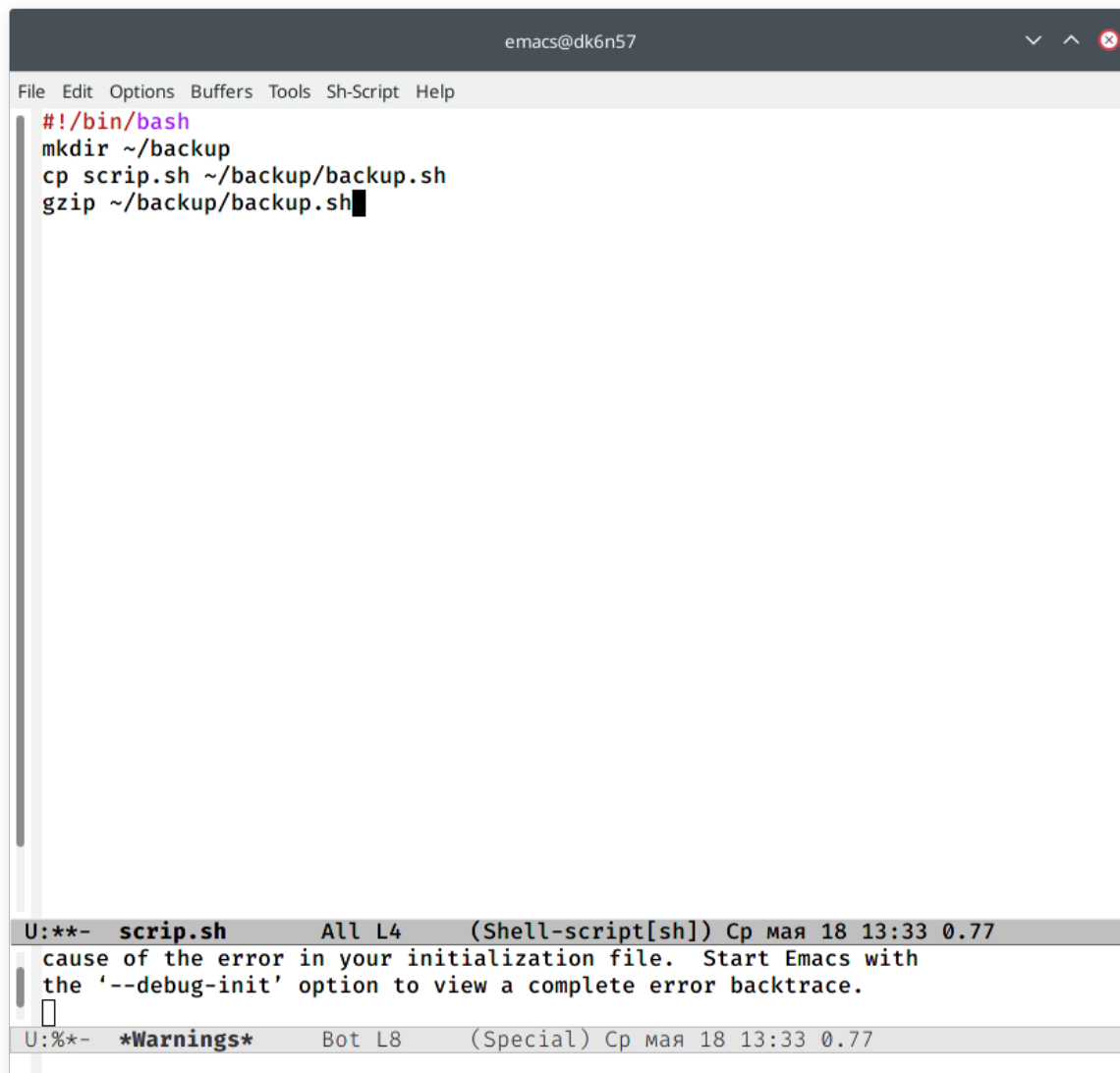


Рис. 3.4: заходим в текстовый редактор

3.5



The screenshot shows an Emacs editor window titled 'emacs@dk6n57'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Sh-Script', and 'Help'. The main text area contains the following shell script code:

```
#!/bin/bash
mkdir ~/backup
cp scrip.sh ~/backup/backup.sh
gzip ~/backup/backup.sh
```

At the bottom of the window, there is a status bar with three entries:

- U:**- scrip.sh** All L4 (Shell-script[sh]) Cp мая 18 13:33 0.77
- cause of the error in your initialization file. Start Emacs with the '--debug-init' option to view a complete error backtrace.
- U:%*- *Warnings*** Bot L8 (Special) Cp мая 18 13:33 0.77

Рис. 3.5: пишем код и сохраняем

3.6

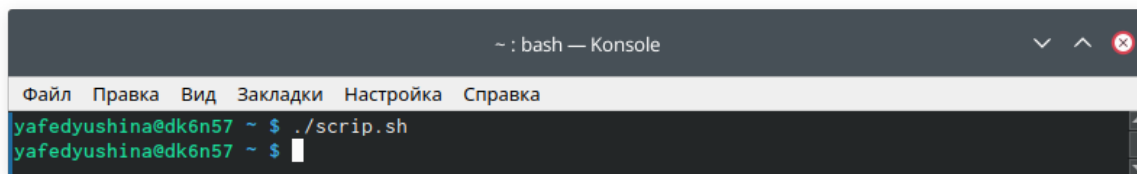


Рис. 3.6: скрипт файл

3.7

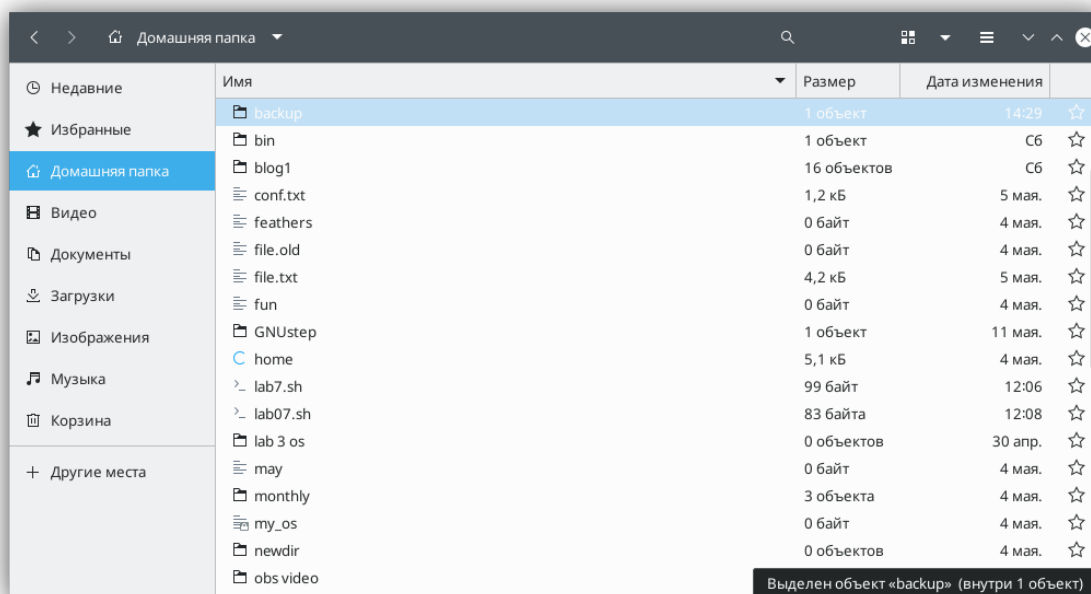


Рис. 3.7: backup

3.8

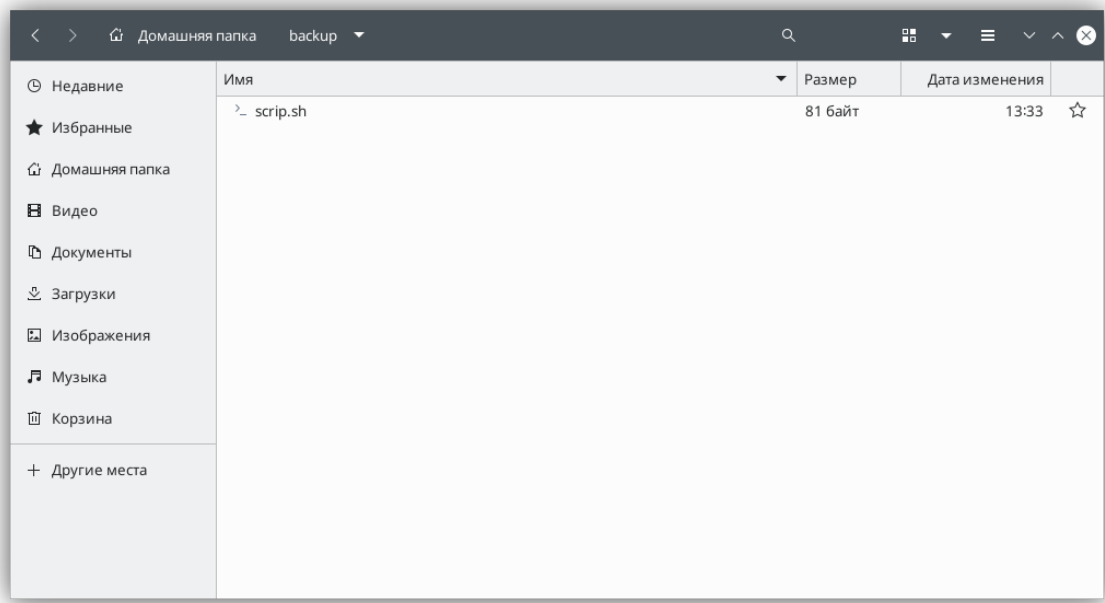


Рис. 3.8: файл

4 Выводы

В ходе выполнения лабораторной работы №10 я изучила основы программирования в оболочке ОС UNIX/Linux и научилась писать небольшие командные файлы.

5 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 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`.
4. 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. 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 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 при входе в систему.

ливается 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 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 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`.