

# **Отчет по лабораторной работе №10**

**дисциплина: операционные системы**

**Шмаков Максим Павлович**

# Содержание

Цель работы	5
Задание	6
Выполнение лабораторной работы	7
Выводы	15
Контрольные вопросы	16

# Список иллюстраций

0.1. рис. 1 . . . . .	7
0.2. рис. 2 . . . . .	8
0.3. рис. 3 . . . . .	8
0.4. рис. 4 . . . . .	9
0.5. рис. 5 . . . . .	9
0.6. рис. 6 . . . . .	10
0.7. рис. 7 . . . . .	10
0.8. рис. 8 . . . . .	10
0.9. рис. 9 . . . . .	11
0.10.рис. 10 . . . . .	11
0.11.рис. 11 . . . . .	12
0.12.рис. 12 . . . . .	12
0.13.рис. 13 . . . . .	13
0.14.рис. 14 . . . . .	13
0.15.рис. 15 . . . . .	14
0.16.рис. 16 . . . . .	14

## **Список таблиц**

# Цель работы

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

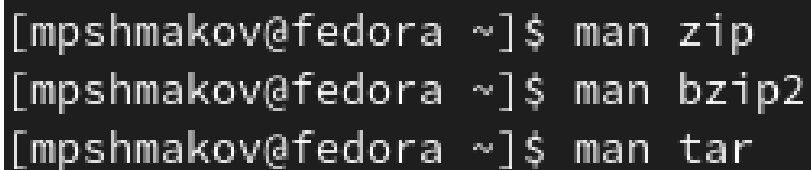
## Задание

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.

# Выполнение лабораторной работы

1. Написать скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию бэкап 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.

Изучил документацию zip, bzip2, tar (рис. [-@fig:001]) (рис. [-@fig:002]) (рис. [-@fig:003]) (рис. [-@fig:004])

A terminal window with a dark background and light gray text. It shows three lines of commands and their outputs: [mpshmakov@fedora ~]\$ man zip, [mpshmakov@fedora ~]\$ man bzip2, and [mpshmakov@fedora ~]\$ man tar. The output of each command is not visible, only the prompt and the command itself.

```
[mpshmakov@fedora ~]$ man zip
[mpshmakov@fedora ~]$ man bzip2
[mpshmakov@fedora ~]$ man tar
```

Рис. 0.1.: рис. 1

```
mpshmakov@fedora:~ — man zip
ZIP(1L) ZIP(1L)
NAME
    zip - package and compress (archive) files
SYNOPSIS
    zip [-aABcdDeEfFghjklLmoqrRSTuvVwXyz!@&] [--longoption ...] [-b path]
        [-n suffixes] [-t date] [-tt date] [zipfile [file ...]] [-xi list]

    zipcloak (see separate man page)

    zipnote (see separate man page)

    zipsplit (see separate man page)

    Note: Command line processing in zip has been changed to support long
    options and handle all options and arguments more consistently. Some
    old command lines that depend on command line inconsistencies may no
    longer work.
DESCRIPTION
    zip is a compression and file packaging utility for Unix, VMS, MSDOS,
    OS/2, Windows 9x/NT/XP, Minix, Atari, Macintosh, Amiga, and Acorn RISC
    OS. It is analogous to a combination of the Unix commands tar(1) and
    compress(1) and is compatible with PKZIP (Phil Katz's ZIP for MSDOS
    systems).
```

Рис. 0.2.: рис. 2

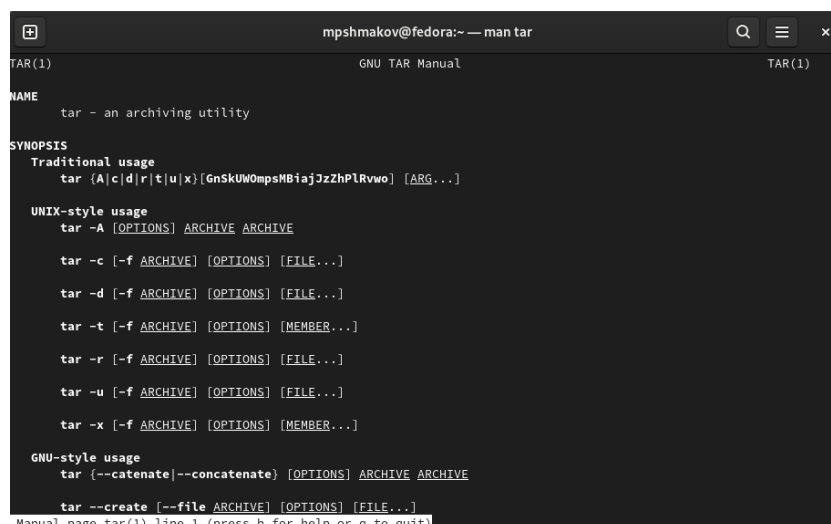
```
mpshmakov@fedora:~ — man bzip2
bzip2(1) General Commands Manual bzip2(1)
NAME
    bzip2, bunzip2 - a block-sorting file compressor, v1.0.8
    bzcat - decompresses files to stdout
    bzip2recover - recovers data from damaged bzip2 files
SYNOPSIS
    bzip2 [-cdfkqstzVL123456789] [filenames ...]
    bunzip2 [-fkvsVL] [filenames ...]
    bzcat [-s] [filenames ...]
    bzip2recover filename
DESCRIPTION
    bzip2 compresses files using the Burrows-Wheeler block sorting text compression algorithm, and Huffman
    coding. Compression is generally considerably better than that achieved by more conventional
    LZ77/LZ78-based compressors, and approaches the performance of the PPM family of statistical compres-
    sors.

    The command-line options are deliberately very similar to those of GNU gzip, but they are not identi-
    cal.

    bzip2 expects a list of file names to accompany the command-line flags. Each file is replaced by a
    compressed version of itself, with the name "original_name.bz2". Each compressed file has the same
    modification date, permissions, and, when possible, ownership as the corresponding original, so that
    these properties can be correctly restored at decompression time. File name handling is naive in the
    sense that there is no mechanism for preserving original file names, permissions, ownerships or dates
    in filesystems which lack these concepts, or have serious file name length restrictions, such as MS-
    Manual page bzip2(1) line 1 (press h for help or q to quit)
```

Рис. 0.3.: рис. 3





```
TAR(1)                                GNU TAR Manual                                TAR(1)

NAME
tar - an archiving utility

SYNOPSIS
Traditional usage
tar {A|c|d|r|t|u|x}[GnSkUW0mpsMBiajJzZhPlRvwo] [ARG...]

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

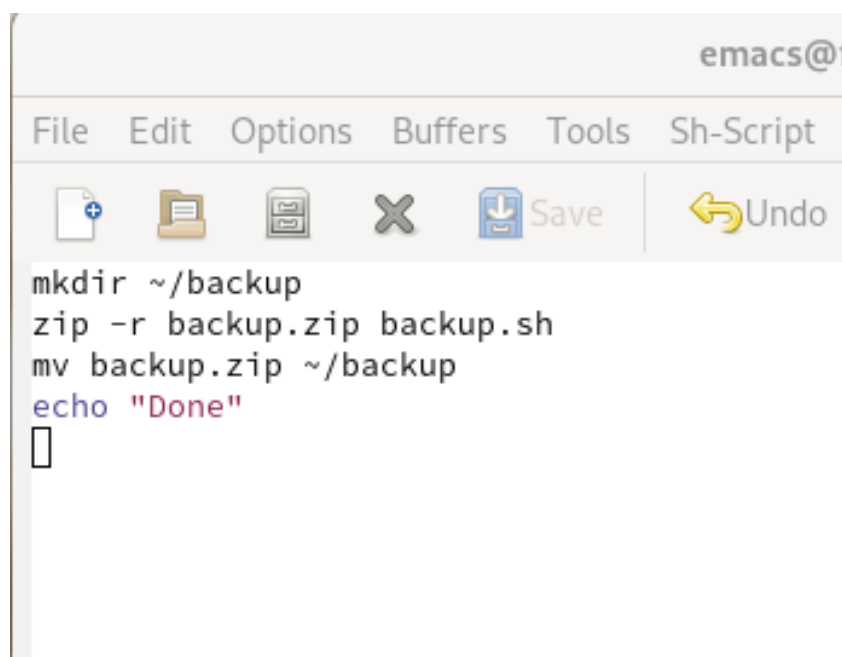
tar -c [-f ARCHIVE] [OPTIONS] [FILE...]
tar -d [-f ARCHIVE] [OPTIONS] [FILE...]
tar -t [-f ARCHIVE] [OPTIONS] [MEMBER...]
tar -r [-f ARCHIVE] [OPTIONS] [FILE...]
tar -u [-f ARCHIVE] [OPTIONS] [FILE...]
tar -x [-f ARCHIVE] [OPTIONS] [MEMBER...]

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

tar --create [--file ARCHIVE] [OPTIONS] [FILE...]
Manual page tar(1) line 1 (press h for help or q to quit)
```

Рис. 0.4.: рис. 4

Создаю файл backup.sh и пишу в нем скрипт. Затем даю права на исполнение этому файлу и тестирую скрипт. Проверяю результат с помощью cd и ls. (рис. [-@fig:005]) (рис. [-@fig:006]) (рис. [-@fig:007])



```
emacs@i

File Edit Options Buffers Tools Sh-Script

[Icons: New, Open, Save, Close, Undo]

mkdir ~/backup
zip -r backup.zip backup.sh
mv backup.zip ~/backup
echo "Done"
█
```

Рис. 0.5.: рис. 5

```
[mpshmakov@fedora ~]$ chmod +x backup.sh
```

Рис. 0.6.: рис. 6

```
[mpshmakov@fedora ~]$ ./backup.sh
  adding: backup.sh (deflated 35%)
Done
[mpshmakov@fedora ~]$ cd backup
[mpshmakov@fedora backup]$ ls
backup.zip
```

Рис. 0.7.: рис. 7

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.

Создаю файл `zadanie2.sh` и пишу в нем скрипт. Даю права на исполнение и проверяю задав больше и меньше 10 аргументов. (рис. [-@fig:008]) (рис. [-@fig:009]) (рис. [-@fig:010])

```
[mpshmakov@fedora ~]$ touch zadanie2.sh
[mpshmakov@fedora ~]$ emacs
```

Рис. 0.8.: рис. 8

```
for num in $@  
do echo $num  
done
```

Рис. 0.9.: рис. 9

```
[mpshmakov@fedora ~]$ ./zadanie2.sh 1 2 3 4  
1  
2  
3  
4  
[mpshmakov@fedora ~]$ ./zadanie2.sh 1 2 3 4 5 6 7 8 9 10 11  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11
```

Рис. 0.10.: рис. 10

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.

Создаю файл `finder.sh` и пишу в нем скрипт. Даю права на исполнение и проверяю работу скрипта. (рис. [-@fig:011]) (рис. [-@fig:012]) (рис. [-@fig:013])

```
[mpshmakov@fedora ~]$ touch finder.sh
[mpshmakov@fedora ~]$ emacs
```

Рис. 0.11.: рис. 11

```
for a in *
do if test -d $a
then echo $a: Директория
else echo -n $a: Файл и
    if test -w $a
    then echo    можно редактировать
    elif test -r $a
    then echo    можно прочитать
    elif test -w $a
    then echo    можно выполнить
    else echo    нельзя ни редактировать, ни прочитать
    fi
fi
done
```

Рис. 0.12.: рис. 12

```
[mpshmakov@fedora ~]$ chmod +x lsanalog.sh
[mpshmakov@fedora ~]$ ./lsanalog.sh
abc1: Файл иможно редактировать
australia: Директория
backup: Директория
backup.sh: Файл иможно редактировать
backup.sh~: Файл иможно редактировать
bin: Директория
conf.txt: Файл иможно редактировать
Desktop: Директория
dfsdfs: Файл иможно редактировать
Documents: Директория
Downloads: Директория
feathers: Файл иможно редактировать
filetest: Директория
filetest.txt: Директория
file.txt: Файл иможно редактировать
index.html: Файл иможно редактировать
index.html.1: Файл иможно редактировать
#lab07.sh#: Файл иможно редактировать
lab07.sh: Файл иможно редактировать
```

Рис. 0.13.: рис. 13

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.

Создаю файл `lsanalog.sh` и пишу в нем скрипт. Даю права на исполнение, проверяю работу скрипта задав разные форматы файлов на вход. (рис. [-@fig:014]) (рис. [-@fig:015]) (рис. [-@fig:016])

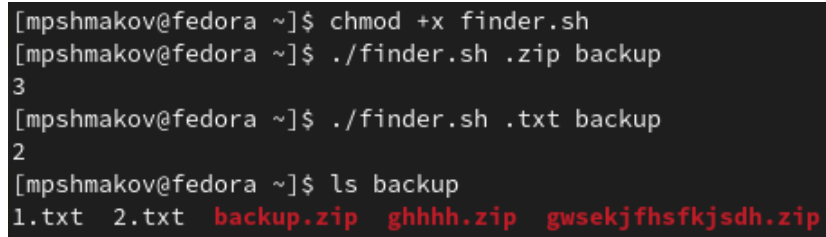
```
[mpshmakov@fedora ~]$ touch lsanalog.sh
[mpshmakov@fedora ~]$ emacs
```

Рис. 0.14.: рис. 14



```
ls $2/*$1 | wc -l
```

Рис. 0.15.: рис. 15



```
[mpshmakov@fedora ~]$ chmod +x finder.sh
[mpshmakov@fedora ~]$ ./finder.sh .zip backup
3
[mpshmakov@fedora ~]$ ./finder.sh .txt backup
2
[mpshmakov@fedora ~]$ ls backup
1.txt 2.txt backup.zip ghhhh.zip gwsekjfhskjsdh.zip
```

Рис. 0.16.: рис. 16

## **Выводы**

В ходе работы я изучил основы программирования в оболочке ОС UNIX/Linux.  
Научиться писать небольшие командные файлы.

# Контрольные вопросы

1. Объясните понятие командной оболочки. Приведите примеры командных оболочек. Чем они отличаются?

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 C-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. Что такое POSIX?

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. Как определяются переменные и массивы в языке программирования bash?

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.

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. Каково назначение операторов `let` и `read`?

Команда `let` является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению.

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. Какие арифметические операции можно применять в языке программирования bash?

В языке программирования `bash` можно применять сложение, умножение, вычитание, деление, нахождение остатка, побитовое дополнение, побитовое сдвигание, сравнение (`>`, `<`, `==`).

6. Что означает операция (( ))?

В них можно записывать условия оболочки `bash`.

7. Какие стандартные имена переменных Вам известны?

`PATH`, `PS1`, `PS2`, `HOME`, `IFS`, `MAIL`, `TERM` и `LOGNAME`.

8. Что такое метасимволы?

Такие символы, как `' < > * ? | " &`, являются метасимволами и имеют для командного процессора специальный смысл.

9. Как экранировать метасимволы?

а. 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 `$`, `'`, `,` и `"`. Например, `- echo *` выведет на экран символ `*`, `- echo ab'|'cd` выведет на экран строку `ab|*cd`.

10. Как создавать и запускать командные файлы?

Создаем текстовый файл, пишем код, даем право на исполнение с помощью команды `chmod +x` (имя файла) и запускаем с помощью `./(имя файла)`

11. Как определяются функции в языке программирования `bash`?

Группу команд можно объединить в функцию. Для этого существует ключевое слово `function`, после которого следует имя функции и список команд, заключённых в фигурные скобки.

12. Каким образом можно выяснить, является файл каталогом или обычным файлом?

С помощью команды test:

– test -f file — истина, если файл file существует; – test -d file — истина, если файл file является каталогом.

13. Каково назначение команд set, typeset и unset?

Для создания массива используется команда set с флагом -A.

typeset используется для объявления и присвоения переменных.

Изъять переменную из программы можно с помощью команды unset

14. Как передаются параметры в командные файлы?

С помощью метасимвола \$. Например, Пусть к командному файлу where имеется доступ по выполнению и этот командный файл содержит следующий конвейер: who | grep \$1. Если Вы введёте с терминала команду where andy, то в случае, если пользователь, зарегистрированный в ОС UNIX под именем andy, в данный момент работает в ОС UNIX, то на терминал будет выведена строка, содержащая номер терминала, используемого указанным пользователем. Если же в данный момент этот пользователь не работает в ОС UNIX, то на терминал ничего не будет выведено.

15. Назовите специальные переменные языка bash и их назначение.

При использовании в командном файле комбинации символов \$# вместо неё будет осуществлена подстановка числа параметров, указанных в командной строке при вызове данного командного файла на выполнение. – \$\* — 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`.