

Отчёт по лабораторной работе № 10

Программирование в командном процессоре ОС UNIX. Командные файлы.

Паулу Антонию Жоау

Содержание

1	Цель работы	4
2	Задание	5
3	Выполнение лабораторной работы	6
3.1	написание программ	6
4	Выводы	12

Список иллюстраций

3.1	Способы архивации	6
3.2	Способы архивации	6
3.3	Способы архивации	7
3.4	Способы архивации	7
3.5	Файл 1	8
3.6	Программа 1	8
3.7	Программа 1	8
3.8	Файл 2	8
3.9	Программа 2	9
3.10	Программа 2	9
3.11	Файл 3	9
3.12	Программа 3	10
3.13	Программа 3	10
3.14	Файл 4	11
3.15	Программа 4	11
3.16	Программа 4	11

1 Цель работы

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

2 Задание

- Ознакомиться с теоретическим материалом.
- Выполнить упражнения.
- Ответить на контрольные вопросы.

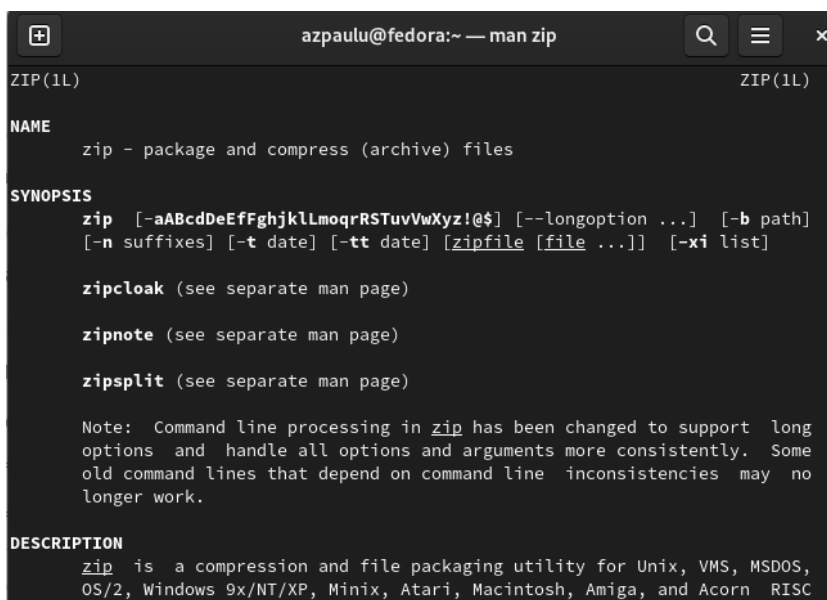
3 Выполнение лабораторной работы

3.1 написание программ

1. Способ использования команд архивации узнали, изучив справку.(рис. [3.1]), (рис. [3.2]), (рис. [3.3]), (рис. [3.4])

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

Рис. 3.1: Способы архивации



```
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
```

Рис. 3.2: Способы архивации

```
azpaulu@fedora:~ — man bzip2
bzip2(1)          General Commands Manual          bzip2(1)

NAME
  bzip2, bunzip2 - a block-sorting file compressor, v1.0.8
  bzip2recover - recovers data from damaged bzip2 files

SYNOPSIS
  bzip2 [ -cdfkqstvwVL123456789 ] [ filenames ... ]
  bunzip2 [ -fkvsVL ] [ 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 compressors.

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

Рис. 3.3: Способы архивации

```
azpaulu@fedora:~ — man tar
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...]
```

Рис. 3.4: Способы архивации

Создали файл. Написали скрипт, который при запуске делает резервную копию самого себя в другую директорию backup в вашем домашнем каталоге. Файл архивируется bzip2. Дали файлу права на исполнение и проверили работу. (рис.

[3.5]), (рис. [3.6]), (рис. [3.7])

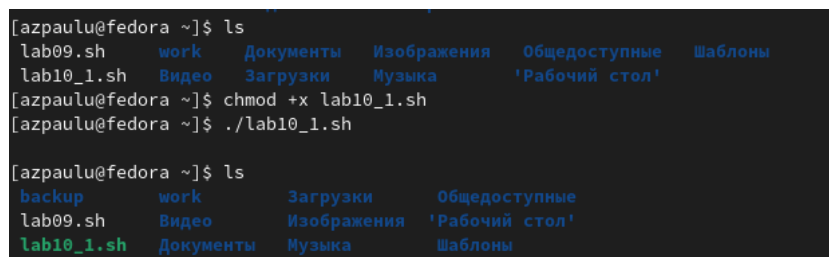
```
[azpaulu@fedora ~]$ touch lab10_1.sh
```

Рис. 3.5: Файл 1



```
#!/bin/bash
name="lab10_1.sh"
mkdir ~/backup/
bzip2 -k ${name}
mv ${name}.bz2 ~/backup/
echo
```

Рис. 3.6: Программа 1



```
[azpaulu@fedora ~]$ ls
lab09.sh  work  Документы  Изображения  Общедоступные  Шаблоны
lab10_1.sh  Видео  Загрузки  Музыка  'Рабочий стол'
[azpaulu@fedora ~]$ chmod +x lab10_1.sh
[azpaulu@fedora ~]$ ./lab10_1.sh

[azpaulu@fedora ~]$ ls
backup  work  Загрузки  Общедоступные
lab09.sh  Видео  Изображения  'Рабочий стол'
lab10_1.sh  Документы  Музыка  Шаблоны
```

Рис. 3.7: Программа 1

2. Создали файл. Написали командный файл, обрабатывающий любое произвольное число аргументов командной строки, в том числе превышающее десять. Скрипт последовательно распечатывает значения всех переданных аргументов. Дали файлу права на исполнение и проверили работу. (рис. [3.8]), (рис. [3.9]), (рис. [3.10])

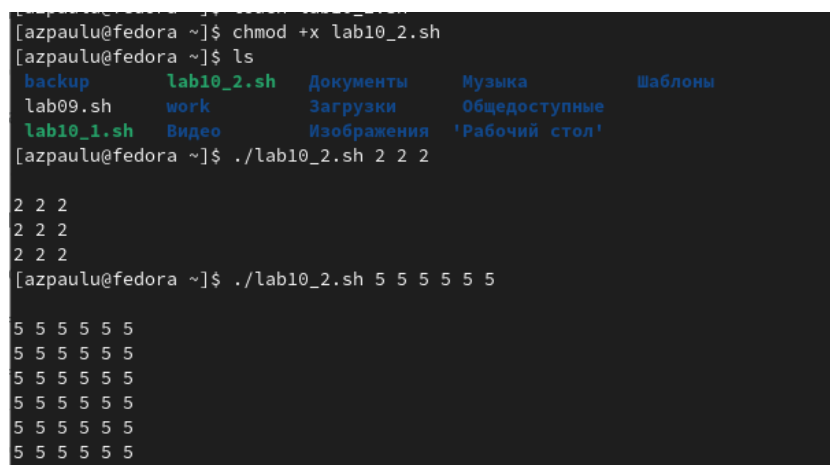
```
[azpaulu@fedora ~]$ touch lab10_2.sh
```

Рис. 3.8: Файл 2



```
Открыть ▾ + lab10_2.sh  
~/  
#!/bin/bash  
echo  
for a in $@  
do echo $a  
done
```

Рис. 3.9: Программа 2



```
[azpaulu@fedora ~]$ chmod +x lab10_2.sh  
[azpaulu@fedora ~]$ ls  
backup lab10_2.sh Документы Музыка Шаблоны  
lab09.sh work Загрузки Общедоступные  
lab10_1.sh Видео Изображения 'Рабочий стол'  
[azpaulu@fedora ~]$ ./lab10_2.sh 2 2 2  
2 2 2  
2 2 2  
2 2 2  
[azpaulu@fedora ~]$ ./lab10_2.sh 5 5 5 5 5 5  
5 5 5 5 5  
5 5 5 5 5  
5 5 5 5 5  
5 5 5 5 5  
5 5 5 5 5  
5 5 5 5 5  
5 5 5 5 5
```

Рис. 3.10: Программа 2

3. Создали файл. Написали командный файл — аналог команды `ls` (без использования самой этой команды и команды `dir`). Он выдаёт информацию о нужном каталоге и выводит информацию о возможностях доступа к файлам этого каталога. Дали файлу права на исполнение и проверили работу. (рис. [3.11]), (рис. [3.12]), (рис. [3.13])



```
[azpaulu@fedora ~]$ touch lab10_3.sh
```

Рис. 3.11: Файл 3

```
Открыть ▾ + lab10_3.sh
~/

#!/bin/bash

a="$1"
for i in ${a}/* do
echo "$1"
if test -f $i
then echo "обычный файл"
fi
if test -d $i
then echo "каталог"
fi
if test -r $i
then echo "чтение разрешено"
fi
if test -w $i
then echo "запись разрешена"
fi
if test -x $i
then echo "выполнение разрешено"
fi
done
```

Рис. 3.12: Программа 3

```
[azpaulu@fedora ~]$ chmod +x lab10_3.sh
[azpaulu@fedora ~]$ ls
backup      lab10_2.sh  work        Загрузки    Общедоступные
lab09.sh    lab10_3.sh  Видео       Изображения 'Рабочий стол'
lab10_1.sh  lab10_3.sh~ Документы   Музыка      Шаблоны
[azpaulu@fedora ~]$ ./lab10_3.sh
/afs
каталог
чтение разрешено
выполнение разрешено
/bin
каталог
чтение разрешено
выполнение разрешено
/boot
каталог
чтение разрешено
выполнение разрешено
/dev
```

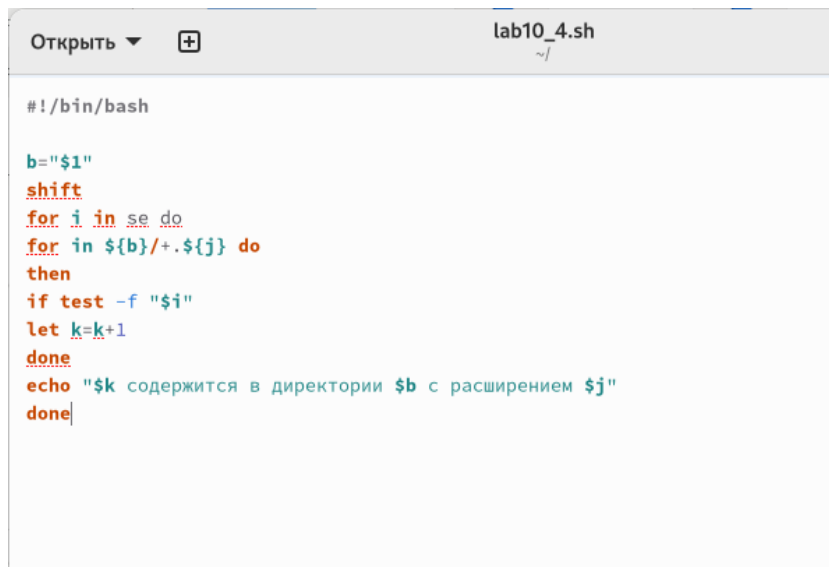
Рис. 3.13: Программа 3

4. Создали файл. Написали 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. Дали файлу права на исполнение и проверили работу. (рис. [3.14]), (рис. [3.15]), (рис.

[3.16])

```
azpaulu@fedora ~$ touch lab10_4.sh
azpaulu@fedora ~$
```

Рис. 3.14: Файл 4

A screenshot of a text editor window titled 'lab10_4.sh'. The editor shows a bash script with the following content:

```
#!/bin/bash

b="$1"
shift
for i in $(ls $b); do
  for j in $(ls $b/$i); do
    if test -f "$j"
    then
      let k=k+1
      echo "$k содержится в директории $b с расширением $j"
    fi
  done
done
```

Рис. 3.15: Программа 4

```
azpaulu@fedora ~$ chmod +x lab10_4.sh
azpaulu@fedora ~$ ./lab10_4.sh ~ sh
5 содержится в директории /home/azpaulu с расширением sh
azpaulu@fedora ~$ ls
backup  lab10_3.sh  work  Изображения  Шаблоны
lab09.sh  lab10_3.sh~  Видео  Музыка
lab10_1.sh  lab10_4.sh  Документы  Общедоступные
lab10_2.sh  lab10_4.sh~  Загрузки  'Рабочий стол'
```

Рис. 3.16: Программа 4

4 Выводы

В ходе выполнения лабораторной работы ознакомились с основами программирования в оболочке ОС UNIX/Linux. Научились писать небольшие командные файлы. # 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: 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 С, но операторы управления программой совместимы с операторами оболочки Борна; BASH – сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек С 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/Linuxподобных 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, 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 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 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. 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 её интерпретацию.
10. 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.
11. 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).

12. 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.
13. 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.

14. 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.