

Лабораторная работа №10

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

Чемоданова Ангелина Александровна

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	7
5	Выводы	12
6	Контрольные вопросы	13

Список иллюстраций

4.1	Справка по zip, создание каталога	7
4.2	Первый скрипт	8
4.3	Работа первого скрипта	8
4.4	Второй скрипт	9
4.5	Работа второго скрипта	9
4.6	Третий скрипт	10
4.7	Работа третьего скрипта	10
4.8	Четвертый скрипт	11
4.9	Работа четвертого скрипта	11

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.

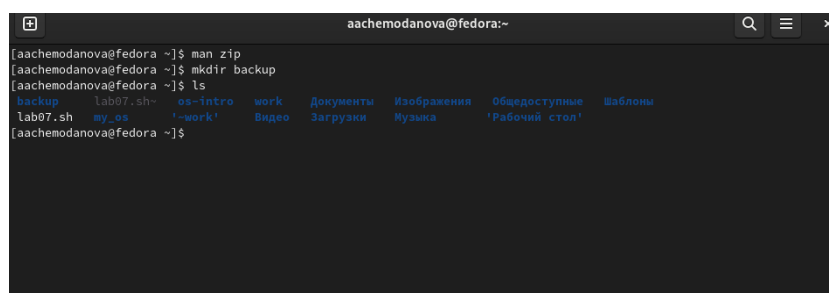
3 Теоретическое введение

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). POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ.

4 Выполнение лабораторной работы

Напишем скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. Способ использования команд архивации узнаем, изучив справку.

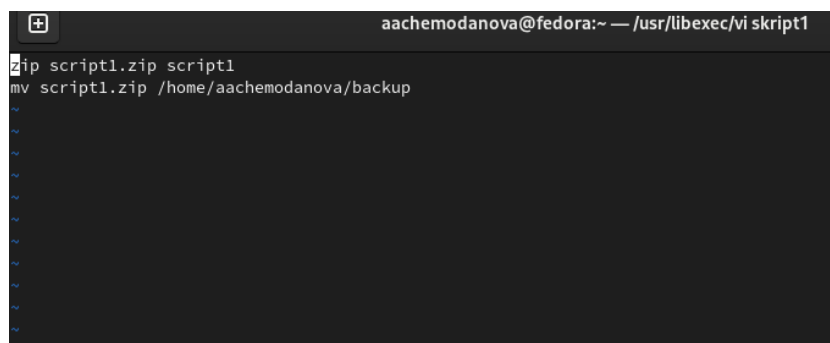
Справка по zip, создание каталога(рис. 4.1).



```
aachemodanova@fedora:~$ man zip
aachemodanova@fedora ~]$ mkdir backup
aachemodanova@fedora ~]$ ls
backup  lab07.sh  os-intro  work  Документы  Изображения  Общедоступные  Шаблоны
lab07.sh  my_os    '-work'   Видео  Загрузки  Музыка      'Рабочий стол'
```

Рис. 4.1: Справка по zip, создание каталога

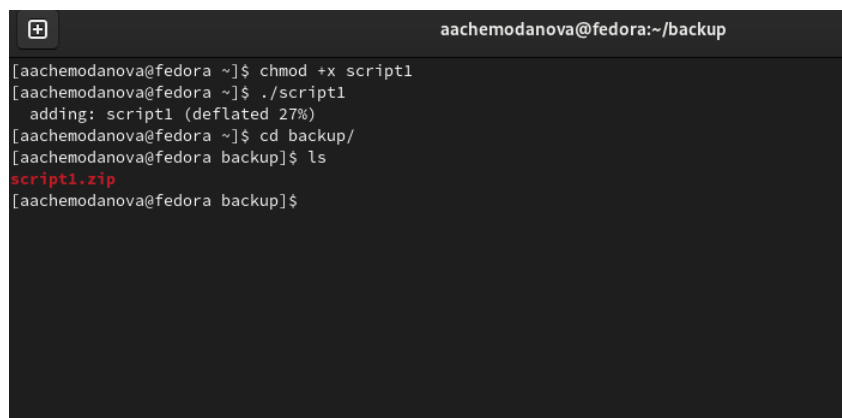
Первый скрипт(рис. 4.2).

A terminal window with a dark background. The title bar shows 'aachemodanova@fedora:~ — /usr/libexec/vi skript1'. The terminal content shows the command 'zip script1.zip script1' being entered, followed by 'mv script1.zip /home/aachemodanova/backup'. Below these, there are several blue tilde characters (~) indicating a continuation of the script or a list of files.

```
aachemodanova@fedora:~ — /usr/libexec/vi skript1
zip script1.zip script1
mv script1.zip /home/aachemodanova/backup
~
~
~
~
~
~
~
~
~
~
```

Рис. 4.2: Первый скрипт

Работа первого скрипта (рис. 4.3).

A terminal window with a dark background. The title bar shows 'aachemodanova@fedora:~/backup'. The terminal content shows the execution of the script: 'chmod +x script1', './script1' (which outputs 'adding: script1 (deflated 27%)'), 'cd backup/', 'ls' (which outputs 'script1.zip'), and finally 'ls' in the home directory.

```
aachemodanova@fedora:~/backup
[aachemodanova@fedora ~]$ chmod +x script1
[aachemodanova@fedora ~]$ ./script1
adding: script1 (deflated 27%)
[aachemodanova@fedora ~]$ cd backup/
[aachemodanova@fedora backup]$ ls
script1.zip
[aachemodanova@fedora backup]$
[aachemodanova@fedora ~]$ ls
```

Рис. 4.3: Работа первого скрипта

Напишем пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять.

Сам скрипт. (рис. 4.4).

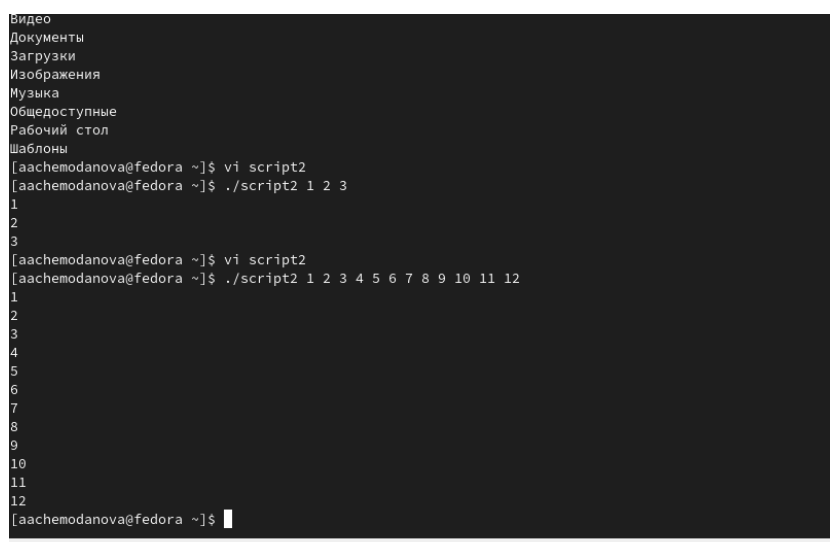


```
aachemodanova@fedora:~ — /u
#!/bin/bash
# echo 'Vvedite'
# read n
for A in $*
do echo $A
done

~
~
~
~
~
~
~
~
~
```

Рис. 4.4: Второй скрипт

Работа второго скрипта (рис. 4.5).



```
видео
Документы
Загрузки
Изображения
Музыка
Общедоступные
Рабочий стол
Шаблоны
[aachemodanova@fedora ~]$ vi script2
[aachemodanova@fedora ~]$ ./script2 1 2 3
1
2
3
[aachemodanova@fedora ~]$ vi script2
[aachemodanova@fedora ~]$ ./script2 1 2 3 4 5 6 7 8 9 10 11 12
1
2
3
4
5
6
7
8
9
10
11
12
[aachemodanova@fedora ~]$
```

Рис. 4.5: Работа второго скрипта

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

Третий скрипт. (рис. 4.6).

```
aachemodanova@fedora:~ — /usr/libexec/vi script3
for A in *
do if test -d $A
then echo $A: is a directory
else echo -n $A: is a file and
if test -w $A
then echo writeable
elif test -r $A
then echo readable
else echo neither readable nor writeable
fi
fi
done
~
~
~
~
~
~
~
~
```

Рис. 4.6: Третий скрипт

Работа третьего скрипта (рис. 4.7).

```
aachemodanova@fedora:~
[aachemodanova@fedora ~]$ vi script3
[aachemodanova@fedora ~]$ chmod +x script3
[aachemodanova@fedora ~]$ ./script3
backup: is a directory
lab07.sh: is a file andwriteable
lab07.sh~: is a file andwriteable
my_os: is a directory
os-intro: is a directory
script1: is a file andwriteable
script2: is a file andwriteable
script3: is a file andwriteable
~work: is a directory
work: is a directory
Видео: is a directory
Документы: is a directory
Загрузки: is a directory
Изображения: is a directory
Музыка: is a directory
Общедоступные: is a directory
./script3: строка 2: test: Рабочий: ожидается бинарный оператор
Рабочий стол: is a file and./script3: строка 5: test: Рабочий: ожидается бинарный оператор
./script3: строка 7: test: Рабочий: ожидается бинарный оператор
neither readable nor writeable
Шаблоны: is a directory
[aachemodanova@fedora ~]$
```

Рис. 4.7: Работа третьего скрипта

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.

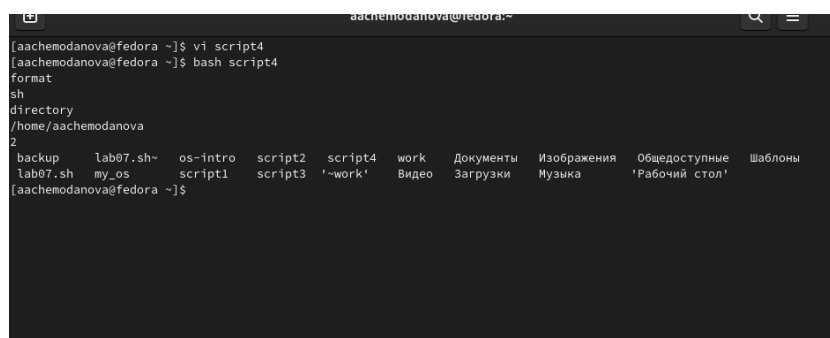
Четвертый скрипт. (рис. 4.8).



```
aachemodanova@fedora:~ — /usr/libexec/vi script4
#!/bin/bash
format=""
directory=""
echo "format"
read format
echo "directory"
read directory
find "${directory}" -name "*.${format}" -type f | wc -l
ls
```

Рис. 4.8: Четвертый скрипт

Работа четвертого скрипта (рис. 4.9).



```
aachemodanova@fedora:~
[aachemodanova@fedora ~]$ vi script4
[aachemodanova@fedora ~]$ bash script4
format
sh
directory
/home/aachemodanova
2
backup  lab07.sh~  os-intro  script2  script4  work  Документы  Изображения  Общедоступные  Шаблоны
lab07.sh  my_os  script1  script3  '~work'  Видео  Загрузки  Музыка  'Рабочий стол'
[aachemodanova@fedora ~]$
```

Рис. 4.9: Работа четвертого скрипта

5 Выводы

Мы изучили основы программирования в оболочке ОС UNIX/Linux. Научились писать небольшие командные файлы.

6 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. Рассмотрим основные элементы программирования в оболочке `bash`. В других оболочках большинство команд будет совпадать с описанными ниже.

3. Как определяются переменные и массивы в языке программирования `bash`?

программирования `bash`? 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 `$`.

Использование значения, присвоенного некоторой переменной, называется подстановкой. Для того чтобы имя переменной не сливалось с символами, которые могут следовать за ним в командной строке, при подстановке в общем случае используется следующая форма записи: `${имя переменной}`. Например, использование команд

```
b=/tmp/andy2
```

```
ls -l myfile > ${b}lsudo apt-get install texlive-luatex
```

приведёт к переназначению стандартного вывода команды `ls` с терминала на файл `/tmp/andy-ls`, а использование команды `ls -l>$bls` приведёт к подстановке в командную строку значения переменной `bls`. Если переменной `bls` не было предварительно присвоено никакого значения, то её значением будет символ пробела.

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

`let` - команда, после которой аргументы представляют собой выражение, подлежащее вычислению.

`read` - команда, позволяющая читать переменные, вводимые с компьютера.

5. Какие арифметические операции можно применять в языке программирования `bash`?

Простейшими математическими выражениями являются сложение (+), вычитание (-), умножение (*), целочисленное деление (/) и целочисленный остаток от деления (%). Однако их намного больше

6. Что означает операция `(())`?

Условия оболочки `bush`.

7. Какие стандартные имена переменных Вам известны?

`PATH`; `PS1`; `PS2`; `HOME`; `IFS`; `MAIL`; `TERM`; `LOGNAME`

8. Что такое метасимволы?

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

9. Как экранировать метасимволы?

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 \$, ' , , "

10. Как создавать и запускать командные файлы?

Нужно просто создать файл через touch, а затем сделать его исполняемым через `chmod +x/название`

11. Как определяются функции в языке программирования bash?

Указать ключевое слово `function`, затем написать её название и открыть фигурную скобку.

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

Командой `ls -lrt`. Есть `d`, то каталог.

13. Каково назначение команд `set`, `typeset` и `unset`?

`set` - установка переменных оболочек и сред

`unset` - удаление переменной из командной оболочки

`typeset` - наложение ограничений на переменные

14. Как передаются параметры в командные файлы?

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. 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.

Вот ещё несколько специальных переменных, используемых в командных файлах:

- $\$*$ — 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;
- $\${name}[n]$ — 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;