

# **Отчет о выполнении лабораторной работы №10**

**Дисциплина: операционные системы**

Астраханцева А. А.

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задание</b>	<b>5</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>6</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
<b>5</b>	<b>Выводы</b>	<b>18</b>

## Список иллюстраций

4.1	Создание файла progr1.sh и каталога backup . . . . .	7
4.2	Текст первой прогаммы . . . . .	7
4.3	Проверка работы программы №1 . . . . .	8
4.4	Создание файла progr2.sh . . . . .	8
4.5	Текст второй прогаммы . . . . .	8
4.6	Проверка работы программы №2 . . . . .	9
4.7	Создание файла progr3.sh . . . . .	9
4.8	Текст третьей прогаммы . . . . .	10
4.9	Проверка работы программы №3 . . . . .	11
4.10	Создание файла progr4.sh . . . . .	11
4.11	Текст третьей прогаммы . . . . .	12
4.12	Проверка работы программы №4 . . . . .	12

# 1 Цель работы

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

## 2 Задание

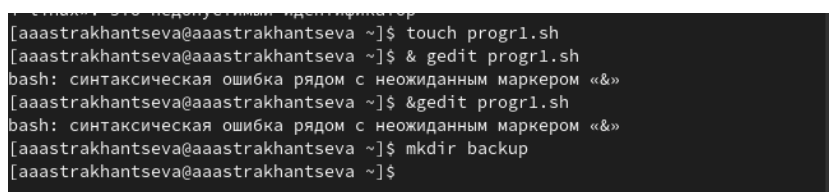
1. Написать скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. Способ использования команд 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 язык shell (в переводе – оболочка) фактически есть язык программирования очень высокого уровня. На этом языке пользователь осуществляет управление компьютером. Обычно, после входа в систему пользователь начинает взаимодействовать с командной оболочкой. Признаком того, что оболочка (shell) готова к приему команд служит выдаваемое ею на экран приглашение (prompt). В простейшем случае это знак доллара (“\$”). Shell не является необходимым и единственным командным языком (хотя именно он стандартизован в рамках POSIX [POSIX 1003.2] - стандарта мобильных систем). Например, немалой популярностью пользуется язык cshell, есть также kshell, bashell (из наиболее популярных в последнее время) и другие. Более того, каждый пользователь может создать свой командный язык. Может одновременно на одном экземпляре операционной системы работать с разными командными языками.

## 4 Выполнение лабораторной работы

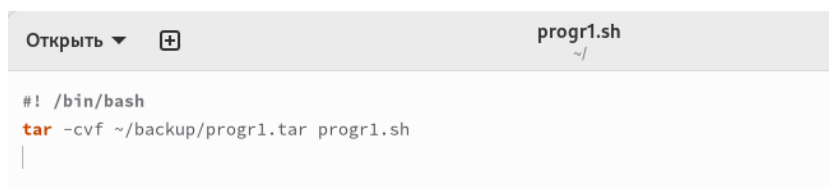
Для начала создаем файл для нашего командного файла с названием `progr1.sh`. И создадим каталог с названием `backup` в домашней директории (рис. 4.1).



```
[aaastrakhantseva@aaastrakhantseva ~]$ touch progr1.sh
[aaastrakhantseva@aaastrakhantseva ~]$ & gedit progr1.sh
bash: синтаксическая ошибка рядом с неожиданным маркером «&»
[aaastrakhantseva@aaastrakhantseva ~]$ &gedit progr1.sh
bash: синтаксическая ошибка рядом с неожиданным маркером «&»
[aaastrakhantseva@aaastrakhantseva ~]$ mkdir backup
[aaastrakhantseva@aaastrakhantseva ~]$
```

Рис. 4.1: Создание файла `progr1.sh` и каталога `backup`

Далее в созданном файле пишем текст программы. С помощью команды `tar` мы создаем сжатую копию файла `progr1.sh` в директории `backup` (рис. 4.2).



```
Открыть ▾ + progr1.sh
~/
#!/bin/bash
tar -cvf ~/backup/progr1.tar progr1.sh
|
```

Рис. 4.2: Текст первой программы

Запускаем файл `progr1.sh` `bash progr1.sh` и проверяем, что все работает корректно. То есть в директории `backup` создается файл `progr1.tar` (рис. 4.3).

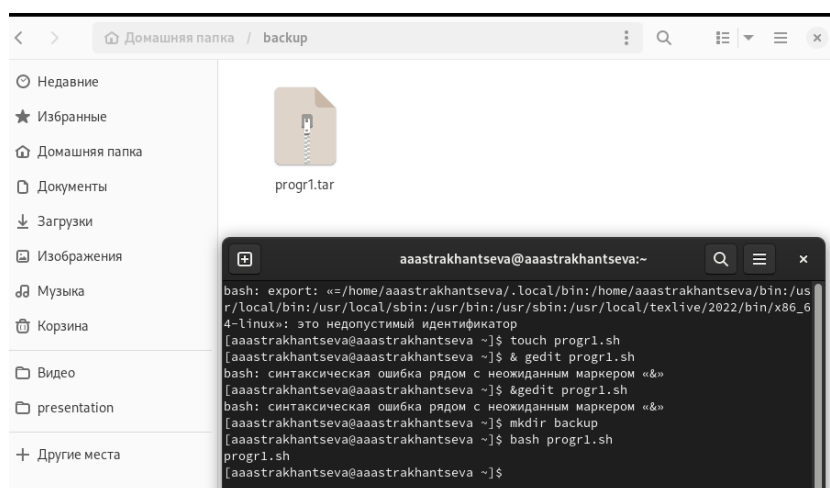


Рис. 4.3: Проверка работы программы №1

Создаем файл для нашего командного файла с названием `progr2.sh` (рис. 4.4).

```
[aaastrakhantseva@aaastrakhantseva ~]$ touch progr2.sh
[aaastrakhantseva@aaastrakhantseva ~]$ ls
backup  '#lab07.sh#'  os-intro  text.txt  Загрузки  Общедоступные
bin     lab07.sh      progr1.sh  work      Изображения  'Рабочий стол'
dir1    lab07.sh~    progr2.sh  Документы  Музыка      Шаблоны
[aaastrakhantseva@aaastrakhantseva ~]$
```

Рис. 4.4: Создание файла `progr2.sh`

Далее в созданном файле пишем текст программы. Используем цикл `for` для перебора всех чисел, введенных в терминал. С помощью команды `echo` мы выведем на экран значение переменной `A` чтобы обозначить, что это не просто буква `A`, а именно переменная, используем `$` (рис. 4.5).

```
#!/bin/bash
#echo 'Введите произвольное количество чисел'
#read A
for A in $*
do echo $A
done
```

Рис. 4.5: Текст второй программы



Запускаем файл `progr2.sh` `bash progr2.sh` и проверяем, что все работает корректно. То есть в терминал выводятся те числа, которые мы ввели после вызова команды (рис. 4.6).

```
[aaastrakhantseva@aaastrakhantseva ~]$ bash progr2.sh 2 4 6 5 2
A
A
A
A
A
[aaastrakhantseva@aaastrakhantseva ~]$ bash progr2.sh 2 4 6 5 2
2
4
6
5
2
[aaastrakhantseva@aaastrakhantseva ~]$
```

Рис. 4.6: Проверка работы программы №2

Создаем файл для нашего командного файла с названием `progr3.sh` (рис. 4.7).

```
[aaastrakhantseva@aaastrakhantseva ~]$ touch progr3.sh
[aaastrakhantseva@aaastrakhantseva ~]$ ls
backup      lab07.sh    progr2.sh   Документы   Общедоступные
bin         lab07.sh~   progr3.sh   Загрузки    'Рабочий стол'
dir1        os-intro    text.txt    Изображения Шаблоны
'lab07.sh#' progr1.sh    work        Музыка
[aaastrakhantseva@aaastrakhantseva ~]$
```

Рис. 4.7: Создание файла `progr3.sh`

Далее в созданном файле пишем текст программы. Используем `test` и конструкцию `if - fi` для проверки того, какие файлы содержатся в данной директории (рис. 4.8).

A screenshot of a code editor window. The title bar at the top shows 'Открыть' (Open) with a dropdown arrow and a plus icon, followed by the filename 'progr3.sh' and the path '~/'. The editor area contains a shell script with the following content:

```
#!/bin/bash
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
        if test -r $A
        then
            echo and readable
        else
            echo neither readable nor writeable
        fi
        fi
    fi
done
```

Рис. 4.8: Текст третьей программы

Запускаем файл `progr3.sh` `bash progr3.sh` и проверяем, что все работает корректно. То есть в терминал выводятся файлы и каталоги, после которых подписано, чем именно они являются (рис. 4.9).

```
[aaastrakhantseva@aaastrakhantseva ~]$ bash progr3.sh
backup: is a directory
bin: is a directory
dir1: is a directory
#lab07.sh#: is a file and writeable
and readable
lab07.sh: is a file and writeable
and readable
lab07.sh~: is a file and writeable
and readable
os-intro: is a directory
progr1.sh: is a file and writeable
and readable
progr2.sh: is a file and writeable
and readable
progr3.sh: is a file and writeable
and readable
text.txt: is a file and writeable
and readable
work: is a directory
Документы: is a directory
Загрузки: is a directory
Изображения: is a directory
Музыка: is a directory
Общедоступные: is a directory
Рабочий стол: is a directory
Шаблоны: is a directory
[aaastrakhantseva@aaastrakhantseva ~]$
```

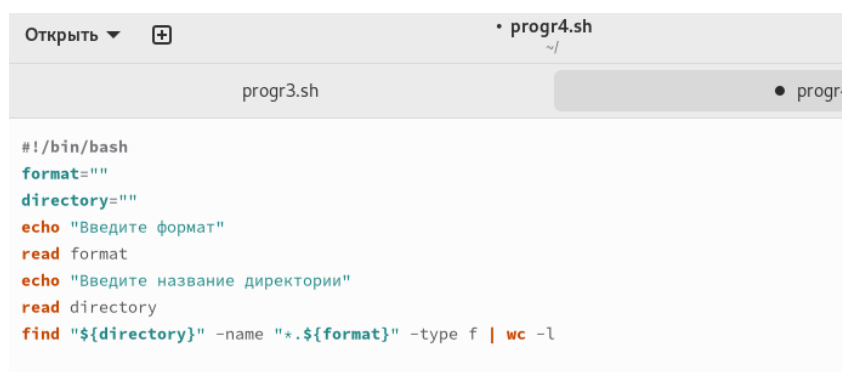
Рис. 4.9: Проверка работы программы №3

Создаем файл для нашего командного файла с названием `progr4.sh` (рис. 4.10).

```
[aaastrakhantseva@aaastrakhantseva ~]$ touch progr4.sh
[aaastrakhantseva@aaastrakhantseva ~]$ ls
backup      lab07.sh    progr2.sh   work        Музыка
bin         lab07.sh~   progr3.sh   Документы   Общедоступные
dir1        os-intro    progr4.sh   Загрузки    'Рабочий стол'
'#lab07.sh#' progr1.sh    text.txt    Изображения Шаблоны
[aaastrakhantseva@aaastrakhantseva ~]$
```

Рис. 4.10: Создание файла `progr4.sh`

Далее в созданном файле пишем текст программы. Принимаем от пользователя название директории и формат файла, после этого с помощью `find` ищем все файлы с заданным расширением в заданной директории и считаем их количество (рис. 4.11).

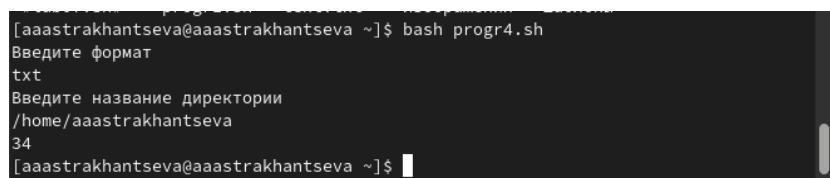


```
Открыть ▾ + • progr4.sh ~/
progr3.sh ● progr4.sh

#!/bin/bash
format=""
directory=""
echo "Введите формат"
read format
echo "Введите название директории"
read directory
find "${directory}" -name ".*${format}" -type f | wc -l
```

Рис. 4.11: Текст третьей программы

Запускаем файл `progr4.sh` `bash progr4.sh` и проверяем, что все работает корректно. То есть в терминал выводятся число с заданным расширением в заданной директории (рис. 4.12).



```
[aaastrakhantseva@aaastrakhantseva ~]$ bash progr4.sh
Введите формат
txt
Введите название директории
/home/aaastrakhantseva
34
[aaastrakhantseva@aaastrakhantseva ~]$
```

Рис. 4.12: Проверка работы программы №4

### Контрольные вопросы

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

Командные процессоры или оболочки - это программы, позволяющие пользователю взаимодействовать с компьютером. Их можно рассматривать как настоящие интерпретируемые языки, которые воспринимают команды пользователя и обрабатывают их. Поэтому командные процессоры также называют интерпретаторами команд. На языках оболочек можно писать программы и выполнять их подобно любым другим программам.

UNIX обладает большим количеством оболочек. Наиболее популярными являются следующие четыре оболочки:

–оболочка Борна (Bourne) - первоначальная командная оболочка UNIX: базовый, но полный набор функций;

–C-оболочка - добавка университета Беркли к коллекции оболочек: она надстраивается над оболочкой Борна, используя C-подобный синтаксис команд, и сохраняет историю выполненных команд;

–оболочка Корна - 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 (англ. Portable Operating System Interface — переносимый интерфейс операционных систем) — набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой (системный API), библиотеку языка C и набор приложений и их интерфейсов. Стандарт создан для обеспечения совместимости различных UNIX-подобных операционных систем и переносимости прикладных программ на уровне исходного кода, но может быть использован и для не-Unix систем.

## 3. Как определяются переменные и массивы в языке 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.

Значение, присвоенное некоторой переменной, может быть впоследствии использовано. Для этого в соответствующем месте командной строки должно быть употреблено имя этой переменной, которому предшествует метасимвол \$.

Для создания массива используется команда set с флагом -A. За флагом следует имя переменной, а затем список значений, разделённых пробелами.

#### 4. Каково назначение операторов let и read?

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

Команда let берет два операнда и присваивает их переменной. Положительным моментом команды let можно считать то, что для идентификации переменной ей не нужен знак доллара; вы можете писать команды типа `let sum=x+7`, и let будет искать переменную x и добавлять к ней 7. Команда let также расширяет другие выражения let, если они заключены в двойные круглые скобки. Таким способом вы можете создавать довольно сложные выражения. Команда let не ограничена простыми арифметическими выражениями.

Команда read позволяет читать значения переменных со стандартного ввода

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

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

#### 6. Что означает операция (( ))?

Условия оболочки bash

#### 7. Какие стандартные имена переменных Вам известны?

Имя переменной (идентификатор) — это строка символов, которая отличает эту переменную от других объектов программы (идентифицирует переменную в программе). При задании имен переменным нужно соблюдать следующие правила: § первым символом имени должна быть буква. Остальные символы — буквы и цифры (прописные и строчные буквы различаются). Можно использовать символ «\_»; § в имени нельзя использовать символ «.»; § число символов в имени не

должно превышать 255; \$ имя переменной не должно совпадать с зарезервированными (служебными) словами языка. Var1, PATH, trash, mon, day, PS1, PS2  
Другие стандартные переменные: –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. Это символы 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. В командном процессоре Си имеется еще несколько стандартных переменных. Значение всех переменных можно просмотреть с помощью команды set.

#### 8. Что такое метасимволы? 9. Как экранировать метасимволы?

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

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 ее интерпретацию.

#### 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. Удалить функцию можно с помощью команды `unset` с флагом `-f`. Команда `typeset` имеет четыре опции для работы с функциями: `-f` — перечисляет определённые на текущий момент функции; `-ft` — при последующем вызове функции иницирует её трассировку; `-fx` — экспортирует все перечисленные функции в любые дочерние программы оболочек; `-fu` — обозначает указанные функции как автоматически загружаемые. Автоматически загружаемые функции хранятся в командных файлах, а при их вызове оболочка просматривает переменную `FPATH`, отыскивая файл с одноимёнными именами функций, загружает его и вызывает эти функции.

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

`ls -lrt` Если есть `d`, то является файл каталогом

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

`set` используется для создания массива, `unset` используется для удаления функции, а `typeset` — для работы с функциями (см. 11 вопрос)

#### 14. Как передаются параметры в командные файлы?



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 нее будет осуществлена подстановка значения параметра с порядковым номером i, т.е. аргумента командного файла с порядковым номером i.

#### 15. Назовите специальные переменные языка bash и их назначение

–  $*$  — отображается вся командная строка или параметры оболочки; –  $?$  — код завершения последней выполненной команды; –  $$$$  — уникальный идентификатор процесса, в рамках которого выполняется команд- ный процессор; –  $!$  — номер процесса, в рамках которого выполняется последняя вызванная на выпол- нение в командном режиме команда; –  $-$  — значение флагов командного процессора; –  $\{#\}$  — *возвращает целое число — количество слов, которые были результатом \$*; –  $\{#name\}$  — возвращает целое значение длины строки в пере-менной 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 пере- менных; –  $\{name:-value\}$  — если значение переменной name не определено, то оно будет заме- нено на указанное value; –  $\{name:value\}$  — проверяется факт существо- вания переменной; –  $\{name=value\}$  — если name не определено, то ему при- сваивается 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 пе- ременная определена, то подставляется value; –  $\{name#pattern\}$  — представ- ляет значение переменной name с удалённым самым коротким левым образцом (pattern); –  $\{#name[*]\}$  и  $\{#name[@]\}$  — эти выражения возвращают количество элементов в массиве name.

## **5 Выводы**

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