

# **Отчет по лабораторной работе №10**

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

Галацан Николай, НПИбд-01-22

# Содержание

|          |                                       |           |
|----------|---------------------------------------|-----------|
| <b>1</b> | <b>Цель работы</b>                    | <b>4</b>  |
| <b>2</b> | <b>Задание</b>                        | <b>5</b>  |
| <b>3</b> | <b>Выполнение лабораторной работы</b> | <b>6</b>  |
| <b>4</b> | <b>Выводы</b>                         | <b>11</b> |
| <b>5</b> | <b>Ответы на контрольные вопросы</b>  | <b>12</b> |

## Список иллюстраций

|     |                                    |    |
|-----|------------------------------------|----|
| 3.1 | Ввод текста программы №1 . . . . . | 6  |
| 3.2 | Запуск программы №1 . . . . .      | 7  |
| 3.3 | Запуск программы №2 . . . . .      | 8  |
| 3.4 | Запуск программы №3 . . . . .      | 9  |
| 3.5 | Запуск программы №4 . . . . .      | 10 |

# 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. 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.

Открываю emacs. Создаю файл lab10\_1.sh, набираю текст программы (рис. 3.1).

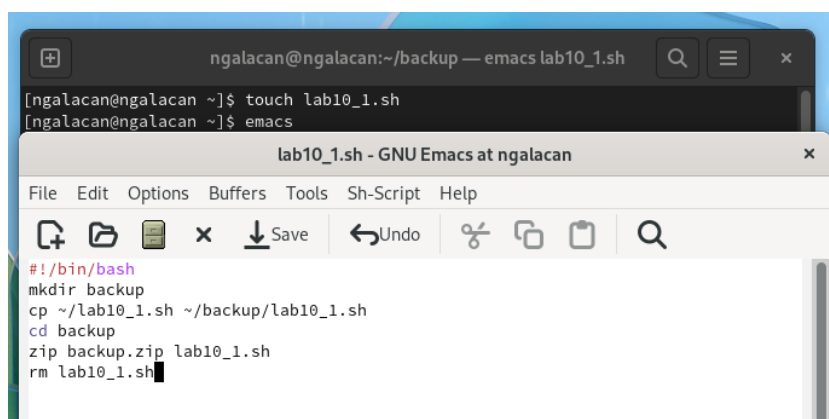


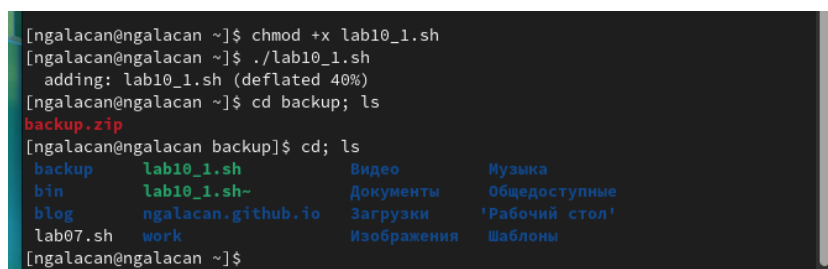
Рис. 3.1: Ввод текста программы №1

*Листинг программы №1:*

```
#!/bin/bash
mkdir backup
cp ~/lab10_1.sh ~/backup/lab10_1.sh
```

```
cd backup
zip backup.zip lab10_1.sh
rm lab10_1.sh
```

Даю файлу право на исполнение и запускаю программу. Скрипт создает директорию backup и копирует в нее исходный код. После этого происходит переход в backup, архивация с помощью zip и удаление исходного файла (рис. 3.2).



```
[ngalacan@ngalacan ~]$ chmod +x lab10_1.sh
[ngalacan@ngalacan ~]$ ./lab10_1.sh
adding: lab10_1.sh (deflated 40%)
[ngalacan@ngalacan ~]$ cd backup; ls
backup.zip
[ngalacan@ngalacan backup]$ cd; ls
backup  lab10_1.sh  Видео  Музыка
bin     lab10_1.sh~  Документы  Общедоступные
blog    ngalacan.github.io  Загрузки  'Рабочий стол'
lab07.sh  work        Изображения  Шаблоны
[ngalacan@ngalacan ~]$
```

Рис. 3.2: Запуск программы №1

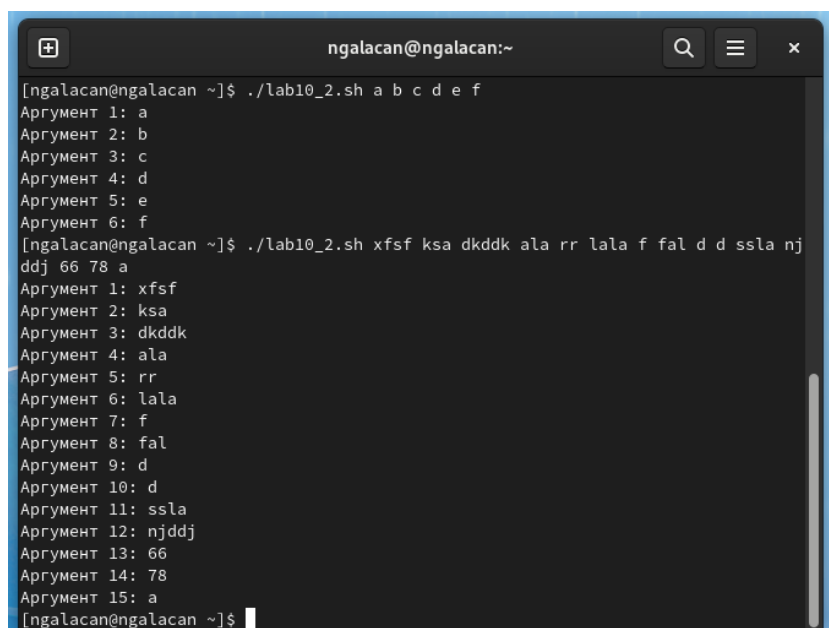
2. Написать пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов.

Создаю файл lab10\_2.sh, набираю текст программы.

*Листинг программы №2:*

```
#!/bin/bash
n=1
while [ -n "$1" ]
do
    echo "Аргумент $n: $1"
    n=$(( $n + 1 ))
    shift
done
```

Даю файлу право на исполнение и запускаю, вводя аргументы в командной строке. Скрипт последовательно выводит на экран введенные аргументы с помощью цикла while (рис. 3.3).



```
ngalacan@ngalacan:~$ ./lab10_2.sh a b c d e f
Аргумент 1: a
Аргумент 2: b
Аргумент 3: c
Аргумент 4: d
Аргумент 5: e
Аргумент 6: f
ngalacan@ngalacan:~$ ./lab10_2.sh xfsf ksa dkddk ala rr lala f fal d d ssla nj
ddj 66 78 a
Аргумент 1: xfsf
Аргумент 2: ksa
Аргумент 3: dkddk
Аргумент 4: ala
Аргумент 5: rr
Аргумент 6: lala
Аргумент 7: f
Аргумент 8: fal
Аргумент 9: d
Аргумент 10: d
Аргумент 11: ssla
Аргумент 12: njddj
Аргумент 13: 66
Аргумент 14: 78
Аргумент 15: a
ngalacan@ngalacan:~$
```

Рис. 3.3: Запуск программы №2

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.

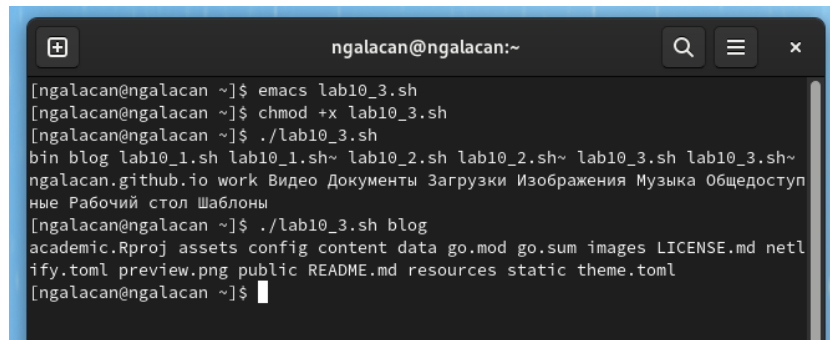
Создаю файл lab10\_3.sh, набираю текст программы.

*Листинг программы №3:*

```
#!/bin/bash
cd $1
echo *
```

Даю файлу право на исполнение и запускаю. Скрипт использует самый простой аналог команды ls: echo \*. При этом путь к директории может передаваться в качестве аргумента (рис. 3.4).



A screenshot of a terminal window with a dark background. The window title is 'ngalacan@ngalacan:~'. The terminal shows the following commands and output:

```
[ngalacan@ngalacan ~]$ emacs lab10_3.sh
[ngalacan@ngalacan ~]$ chmod +x lab10_3.sh
[ngalacan@ngalacan ~]$ ./lab10_3.sh
bin blog lab10_1.sh lab10_1.sh~ lab10_2.sh lab10_2.sh~ lab10_3.sh lab10_3.sh~
ngalacan.github.io work Видео Документы Загрузки Изображения Музыка Общедоступ
ные Рабочий стол Шаблоны
[ngalacan@ngalacan ~]$ ./lab10_3.sh blog
academic.Rproj assets config content data go.mod go.sum images LICENSE.md netl
ify.toml preview.png public README.md resources static theme.toml
[ngalacan@ngalacan ~]$
```

Рис. 3.4: Запуск программы №3

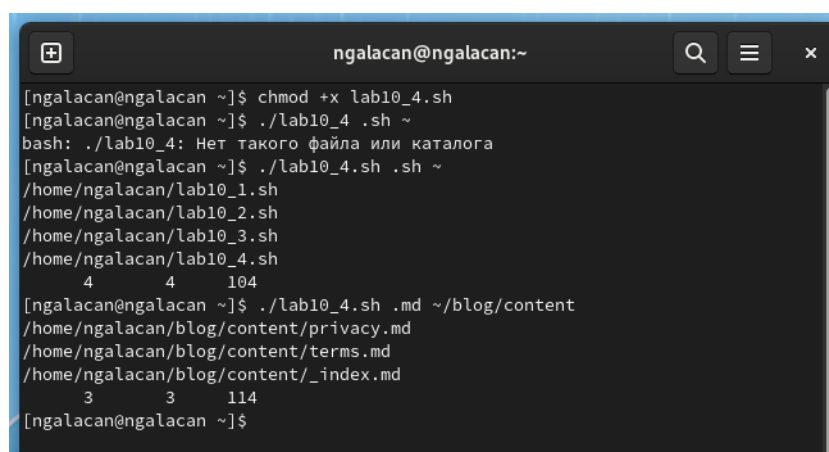
4. Написать командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки.

Создаю файл lab10\_4.sh, набираю текст программы.

*Листинг программы №4:*

```
#!/bin/bash
find $2 -maxdepth 1 -name "$*$1"
find $2 -maxdepth 1 -name "$*$1" | wc
```

Даю файлу право на исполнение и запускаю. В скрипте используется команда find для поиска файлов указанного формата в указанной директории. Во второй раз вывод команды find передается утилите wc, которая подсчитывает количество найденных файлов (рис. 3.5).

A terminal window titled 'ngalacan@ngalacan:~' with search, menu, and close icons in the title bar. The terminal shows the following commands and output:

```
[ngalacan@ngalacan ~]$ chmod +x lab10_4.sh
[ngalacan@ngalacan ~]$ ./lab10_4 .sh ~
bash: ./lab10_4: Нет такого файла или каталога
[ngalacan@ngalacan ~]$ ./lab10_4.sh .sh ~
/home/ngalacan/lab10_1.sh
/home/ngalacan/lab10_2.sh
/home/ngalacan/lab10_3.sh
/home/ngalacan/lab10_4.sh
4      4      104
[ngalacan@ngalacan ~]$ ./lab10_4.sh .md ~/blog/content
/home/ngalacan/blog/content/privacy.md
/home/ngalacan/blog/content/terms.md
/home/ngalacan/blog/content/_index.md
3      3      114
[ngalacan@ngalacan ~]$
```

Рис. 3.5: Запуск программы №4

## **4 Выводы**

Изучены основы программирования в оболочке ОС UNIX/Linux. Приобретен навык написания небольших командных файлов.

## 5 Ответы на контрольные вопросы

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

Командный процессор это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек: 1. оболочка Борна (Bourne shell или sh) это стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций; 2. C оболочка (или csh) это надстройка на оболочкой Борна, использующая Сподобный синтаксис команд с возможностью сохранения истории выполнения команд; 3. Оболочка Корна (или ksh) напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна; 4. BASH сокращение от BourneAgainShell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании FreeSoftwareFoundation).

2. Что такое POSIX?

POSIX (Portable Operating System Interface for Computer Environments ) это 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?

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

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 month day trash`». В переменные `month` 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 такие арифметические операции как сложение (+), вычитание (-), умножение (\*), целочисленное деление (/) и целочисленный остаток от деления (%).

6. Что означает операция (( ))?

В (( )) можно записывать условия оболочки `bash`, а также внутри двойных скобок можно вычислять арифметические выражения и возвращать результат.

7. Какие стандартные имена переменных Вам известны?

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 (`newline`).

- 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 Youhavemail(у Вас есть почта).
- TERM: тип используемого терминала.
- LOGNAME: содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему.

#### 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 мета символа. Экранирование может быть осуществлено с помощью предшествующего мета символу символа , который, в свою очередь, является мета символом. 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. 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 осуществить её 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`.

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

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

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. 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.

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; - \$# возвращает целое число количеств 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.