

Лабораторная работа № 10

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

Бекназарова Виктория Тиграновна

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	9
4	Контрольные вопросы	10

Список иллюстраций

2.1	Задание 1	6
2.2	Задание 2	7
2.3	Задание 3	7
2.4	Задание 4	8

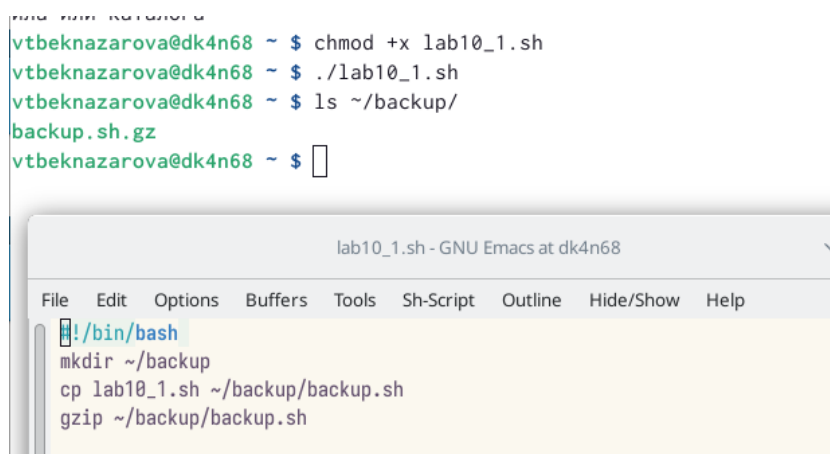
Список таблиц

1 Цель работы

Изучим основы программирования в оболочке ОС UNIX/Linux. Научимся писать небольшие командные файлы.

2 Выполнение лабораторной работы

1. 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 нашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. Способ использования команд архивации необходимо узнать, изучив справку (рис. 2.1).



```
vtbeknazarova@dk4n68 ~ $ chmod +x lab10_1.sh
vtbeknazarova@dk4n68 ~ $ ./lab10_1.sh
vtbeknazarova@dk4n68 ~ $ ls ~/backup/
backup.sh.gz
vtbeknazarova@dk4n68 ~ $
```

The image shows a terminal window with the following commands and output:

```
vtbeknazarova@dk4n68 ~ $ chmod +x lab10_1.sh
vtbeknazarova@dk4n68 ~ $ ./lab10_1.sh
vtbeknazarova@dk4n68 ~ $ ls ~/backup/
backup.sh.gz
vtbeknazarova@dk4n68 ~ $
```

Below the terminal window is a screenshot of the GNU Emacs editor window titled "lab10_1.sh - GNU Emacs at dk4n68". The editor shows the following content:

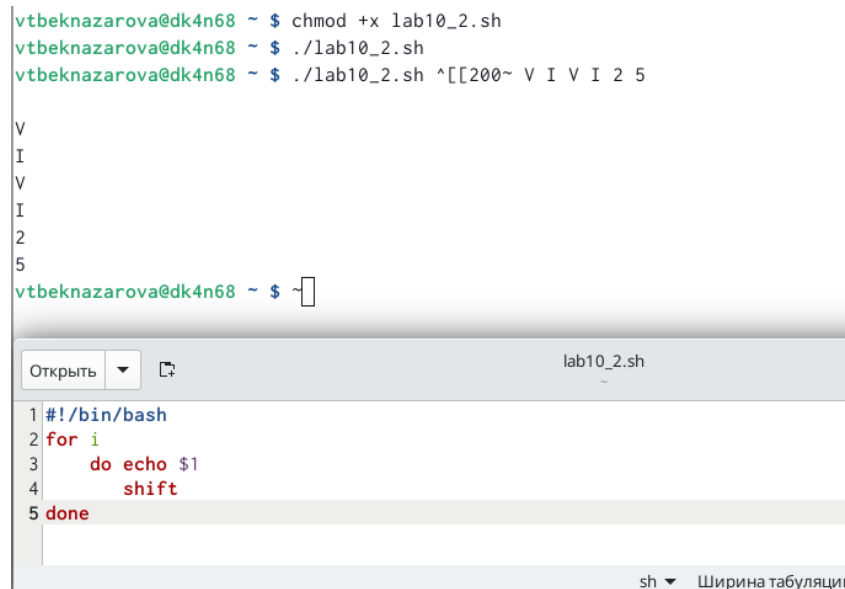
```
#!/bin/bash
mkdir ~/backup
cp lab10_1.sh ~/backup/backup.sh
gzip ~/backup/backup.sh
```

Рис. 2.1: Задание 1

2. Напишем пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов(рис. 2.2).

```
vtbknazarova@dk4n68 ~ $ chmod +x lab10_2.sh
vtbknazarova@dk4n68 ~ $ ./lab10_2.sh
vtbknazarova@dk4n68 ~ $ ./lab10_2.sh ^[[200~ V I V I 2 5

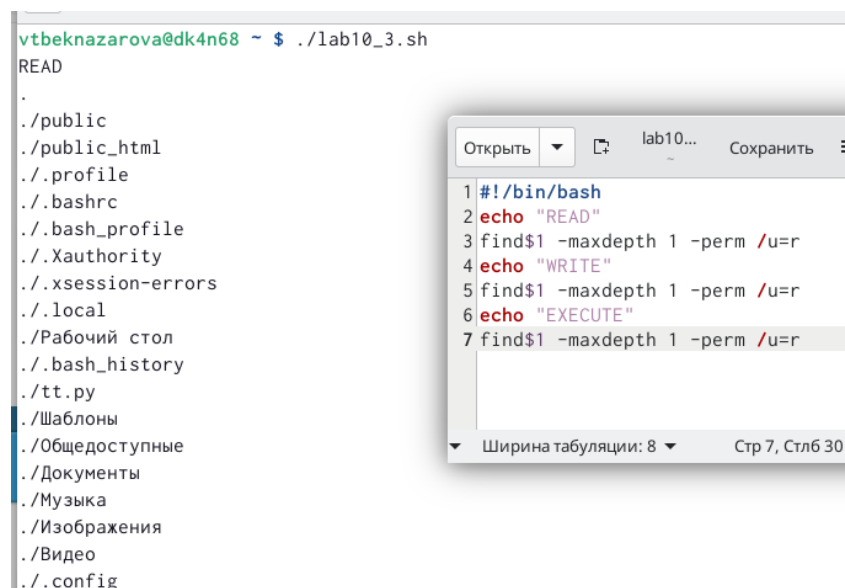
V
I
V
I
2
5
vtbknazarova@dk4n68 ~ $
```



```
1 #!/bin/bash
2 for i
3 do echo $1
4 shift
5 done
```

Рис. 2.2: Задание 2

3. Напишем командный файл — аналог команды ls (без использования самой этой ко- манды и команды dir). Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога(рис. 2.3).



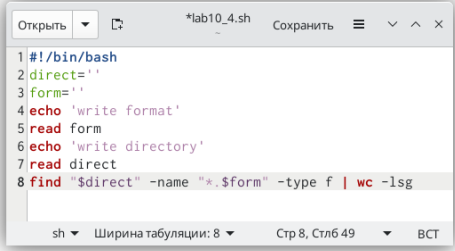
```
vtbknazarova@dk4n68 ~ $ ./lab10_3.sh
READ
.
./public
./public_html
./profile
./bashrc
./bash_profile
./Xauthority
./xsession-errors
./local
./Рабочий стол
./bash_history
./tt.py
./Шаблоны
./Общедоступные
./Документы
./Музыка
./Изображения
./Видео
./config
```

```
1 #!/bin/bash
2 echo "READ"
3 find$1 -maxdepth 1 -perm /u=r
4 echo "WRITE"
5 find$1 -maxdepth 1 -perm /u=r
6 echo "EXECUTE"
7 find$1 -maxdepth 1 -perm /u=r
```

Рис. 2.3: Задание 3

4. 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.(рис. 2.4).

```
./прол.py
./backup
./Untitled8 (1).py
./lab10_2.sh
./lab10_3.sh
vtbknazarova@dk4n68 ~ $ touch lab10_4.sh
vtbknazarova@dk4n68 ~ $ chmod +x lab10_4.sh
vtbknazarova@dk4n68 ~ $ ./lab10_4.sh
write format
sh
write directory
backup
0
```



```
*lab10_4.sh
1 #!/bin/bash
2 direct=''
3 form=''
4 echo 'write format'
5 read form
6 echo 'write directory'
7 read direct
8 find "$direct" -name "$form" -type f | wc -lsg
```

Рис. 2.4: Задание 4

3 Выводы

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

4 Контрольные вопросы

1. Объясните понятие командной оболочки. Приведите примеры командных оболочек. Чем они отличаются? Ответ:
 - a) sh — стандартная командная оболочка UNIX/Linux, содержащая базовый, полный набор функций
 - b) csh — использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд
 - c) ksh — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна
 - d) bash — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна
2. Что такое POSIX? Ответ: POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ.
3. Как определяются переменные и массивы в языке программирования bash? Ответ: Переменные вызываются \$var, где var=чему-то, указанному пользователем, неважно что бы то не было, название файла, каталога или еще чего. Для массивов используется команда set -A
4. Каково назначение операторов let и read? Ответ: let — вычисляет далее заданное математическое значение read — позволяет читать значения переменных со стандартного ввода

5. Какие арифметические операции можно применять в языке программирования `bash`? Ответ: Прибавление, умножение, вычисление, деление), сравнение значений, экспонирование и др.
6. Что означает операция `(())`? Ответ: Это обозначение используется для облегчения программирования для условий `bash`
7. Какие стандартные имена переменных Вам известны? Ответ: Нам известны `HOME`, `PATH`, `BASH`, `ENV`, `PWD`, `UID`, `OLDPWD`, `PPID`, `GROUPS`, `OSTYPE`, `PS1` - `PS4`, `LANG`, `HOSTFILE`, `MAIL`, `TERM`, `LOGNAME`, `USERNAME`, `IFS` и др.
8. Что такое метасимволы? Ответ: Метасимволы это специальные знаки, которые могут использоваться для сокращения пути, поиска объекта по расширению, перед переменными, например «\$» или «*» .
9. Как экранировать метасимволы? Ответ: Добавить перед метасимволом метасимвол «\»
10. Как создавать и запускать командные файлы? Ответ: При помощи команды `chmod`. Надо дать права на запуск `chmod +x название файла`, затем запустить `bash ./название файла` Например у нас файл `lab` Пишем: `chmod +x lab ./lab`
11. Как определяются функции в языке программирования `bash`? Ответ: Объединяя несколько команд с помощью `function`
12. Каким образом можно выяснить, является файл каталогом или обычным файлом? Ответ: Можно задать команду на проверку директория ли это `test -d директория`
13. Каково назначение команд `set`, `typeset` и `unset`? Ответ: `Set` — используется для создания массивов `Unset` — используется для изъятия переменной `Typeset` — используется для присваивания каких-либо функций
14. Как передаются параметры в командные файлы? Ответ: Добавлением аргументов после команды запуска `bash` скрипта

15. Назовите специальные переменные языка bash и их назначение. Ответ:

- `$*` – отображается вся командная строка или параметры оболочки;
- `$?` – код завершения последней выполненной команды;
- `$$` – уникальный идентификатор процесса, в рамках которого выполняется команда;
- `$!` – номер процесса, в рамках которого выполняется последняя вызванная на выполнение команда;
- `$-` – значение флагов командного процессора;
- `${#*}` – возвращает целое число – количество слов, которые были результатом выполнения `$*`;
- `${#name}` – возвращает целое значение длины строки в переменной `name`;
- `${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;
- `${name:-value}` – если значение переменной `name` не определено, то оно будет заменено на `value`;
- `${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 сообщение об ошибке;
- `${name+value}` – это выражение работает противоположно `${name-value}`. Если переменная определена, то подставляется `value`;
- `${name#pattern}` – представляет значение переменной `name` с удалённым самым коротким началом, соответствующим `pattern`;
- `${#name[*]}` и `${#name[@]}` – эти выражения возвращают количество элементов в массиве `name`.