

# Отчёт по лабораторной работе 12

## Программирование в командном процессоре ОС UNIX. Командные файлы

Чуева Злата Станиславовна

### Содержание

Цель работы.....	1
Задание.....	1
Теоретическое введение .....	1
Выполнение лабораторной работы .....	2
Выводы.....	2
Контрольные вопросы.....	2
Список литературы.....	4

### Цель работы

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

### Задание

1. Написать скрипт, который при запуске будет делать резервную копию самого себя.
2. Написать пример командного файла, обрабатывающего любое произвольное число аргументов командной строки.
3. . Написать командный файл — аналог команды ls.
4. Написать командный файл, который получает в качестве аргумента командной строки формат файла.

### Теоретическое введение

При перечислении имён файлов текущего каталога можно использовать следующие символы: – \* — соответствует произвольной, в том числе и пустой строке; – ? — соответствует любому одинарному символу; – [c1-c1] — соответствует любому символу, лексикографически находящемуся между символами c1 и c2. Например, – echo \* — выведет имена всех файлов текущего каталога, что представляет собой простейший аналог команды ls; – ls \*.c — выведет все файлы с последними двумя символами, совпадающими с .c. – echo

prog.? — выведет все файлы, состоящие из пяти или шести символов, первыми пятью символами которых являются prog.. — [a-z]\* — соответствует произвольному имени файла в текущем каталоге, начинающемуся с любой строчной буквы латинского алфавита. 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 экранирова.

## Выполнение лабораторной работы

1.1. Создать файл backup\_script.sh и добавить следующий код:

```
#!/bin/bash

SCRIPT_NAME=$(basename "$0")

BACKUP_DIR="$HOME/backup"

mkdir -p "$BACKUP_DIR"

zip "$BACKUP_DIR/${SCRIPT_NAME}.zip" "$0"

echo "Резервная копия скрипта $SCRIPT_NAME сохранена в $BACKUP_DIR"
```

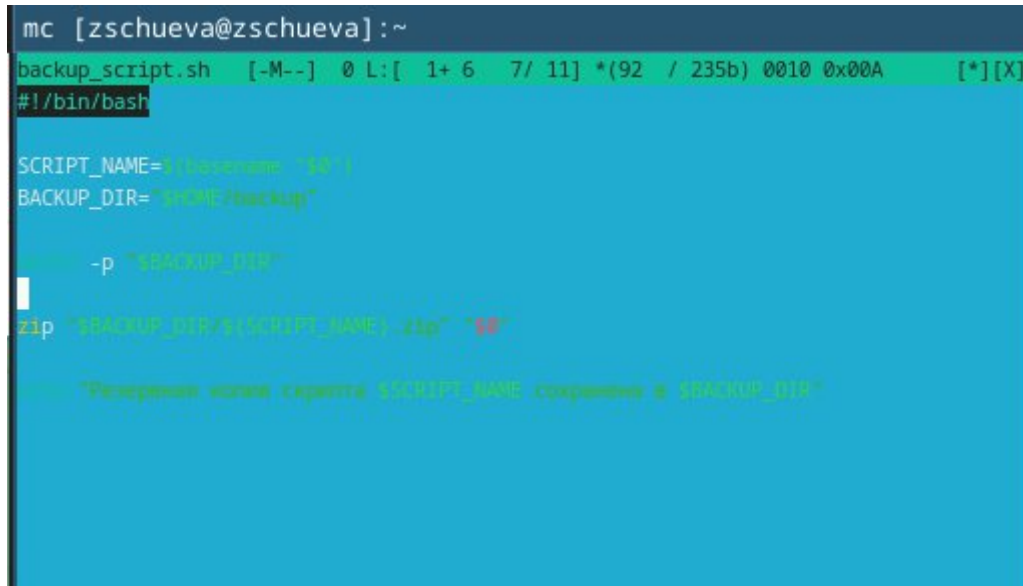


Рис 1.1

1.2. Сделать скрипт исполняемым командой `chmod +x backup_script.sh`.

```
foot
[zschieva@zschieva ~]$ touch backup_script.sh
[zschieva@zschieva ~]$ mc

[zschieva@zschieva ~]$ chmod +x backup_script.sh
[zschieva@zschieva ~]$
```

Рис 1.2

2.1. Создать файл print\_args.sh и добавить следующий код:

```
#!/bin/bash

if [ "$#" -eq 0 ]; then
    echo "Нет переданных аргументов."
    exit 1
fi

for arg in "$@"; do
    echo "$arg"
done
```

```
mc [zschieva@zschieva]:~
print_args.sh [-M--] 0 L:[ 1+ 6 7/ 11] *(112 / 154b) 0010 0x00A [*] [X]
#!/bin/bash

if [ "$#" -eq 0 ]; then
    echo "Нет переданных аргументов."
    exit 1
fi

for arg in "$@"; do
    echo "$arg"
done
```

Рис 2.1

2.2. Сделать скрипт исполняемым командой `chmod +x print_args.sh`

```
[zschieva@zschieva ~]$ touch print_args.sh
[zschieva@zschieva ~]$ mc

[zschieva@zschieva ~]$ chmod +x print_args.sh
[zschieva@zschieva ~]$
```

Рис 2.2

3.1. Создать файл my\_ls.sh и добавить следующий код:

```
#!/bin/bash

if [ -z "$1" ]; then
    echo "Укажите путь к каталогу."
    exit 1
fi

DIRECTORY="$1"

if [ ! -d "$DIRECTORY" ]; then
    echo "Каталог $DIRECTORY не найден."
    exit 1
fi

for file in "$DIRECTORY"/*; do
    if [ -e "$file" ]; then
        PERMISSIONS=$(ls -ld "$file" | awk '{print $1}')
        FILENAME=$(basename "$file")
        echo "$PERMISSIONS $FILENAME"
    fi
done
```

```
mc [zschueva@zschueva]:~
my_ls.sh [-M--] 0 L: [ 1+13 14/ 22] *(222 / 426b) 0010 0x00A [*][X]
#!/bin/bash

if [ -z "$1" ]; then
    echo "Укажите путь к каталогу."
    exit 1
fi

DIRECTORY="$1"

if [ ! -d "$DIRECTORY" ]; then
    echo "Каталог $DIRECTORY не найден."
    exit 1
fi

for file in "$DIRECTORY"/*; do
    if [ -e "$file" ]; then
        PERMISSIONS=$(ls -ld "$file" | awk '{print $1}')
        FILENAME=$(basename "$file")
        echo "$PERMISSIONS $FILENAME"
    fi
done
```

Рис 3.1

3.2. Сделать скрипт исполняемым командой `chmod +x my_ls.sh`

```
[zschueva@zschueva ~]$ touch my_ls.sh
[zschueva@zschueva ~]$ mc

[zschueva@zschueva ~]$ chmod +x my_ls.sh
[zschueva@zschueva ~]$
```

Рис 3.2

4.1. Создать файл `count_files.sh` и добавить следующий код:

```
#!/bin/bash
```

```
# Проверяем, переданы ли аргументы
```

```
if [ "$#" -ne 2 ]; then
```

```
    echo "Использование: $0 <путь к директории> <расширение>"
```

```
    exit 1
```

```
fi
```

```
DIRECTORY="$1"
```

```
EXTENSION="$2"
```

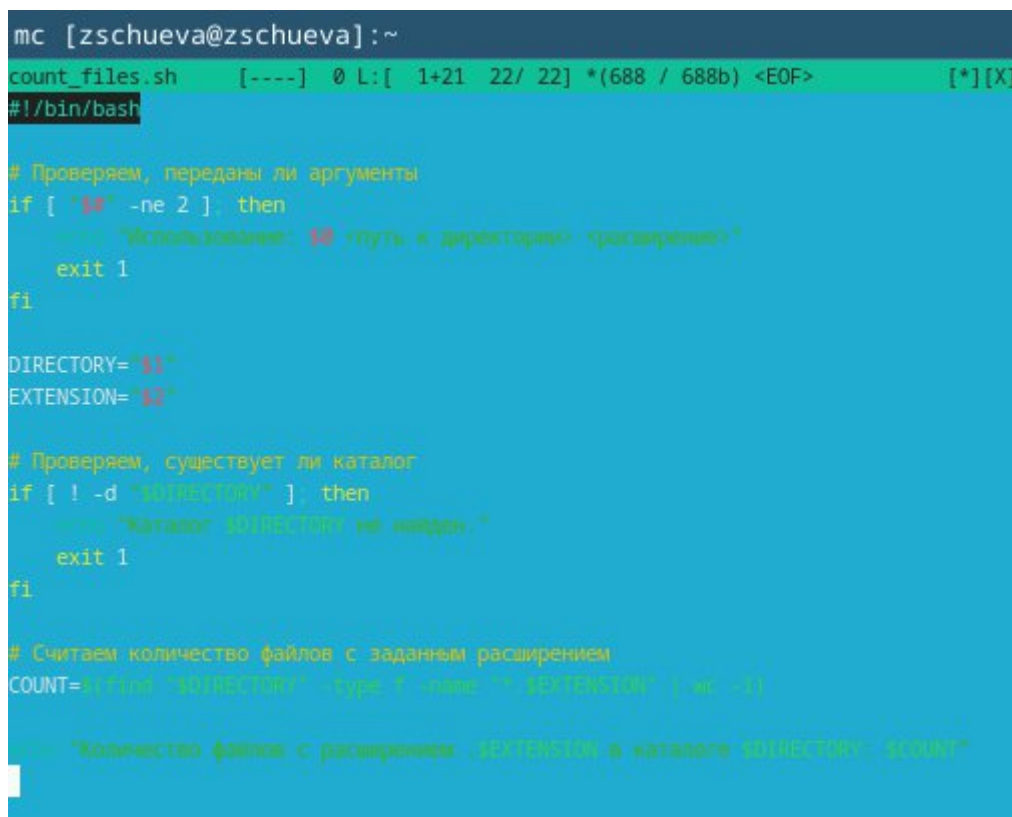
```

# Проверяем, существует ли каталог
if [ ! -d "$DIRECTORY" ]; then
    echo "Каталог $DIRECTORY не найден."
    exit 1
fi

# Считаем количество файлов с заданным расширением
COUNT=$(find "$DIRECTORY" -type f -name ".*$EXTENSION" | wc -l)

echo "Количество файлов с расширением .$EXTENSION в каталоге $DIRECTORY:
$COUNT"

```



```

mc [zschueva@zschueva]:~
count_files.sh  [-----]  0 L:[ 1+21 22/ 22] *(688 / 688b) <EOF>  [*][X]
#!/bin/bash

# Проверяем, переданы ли аргументы
if [ "$#" -ne 2 ]; then
    echo "Использование: $0 <путь к директории> <расширение>"
    exit 1
fi

DIRECTORY="$1"
EXTENSION="$2"

# Проверяем, существует ли каталог
if [ ! -d "$DIRECTORY" ]; then
    echo "Каталог $DIRECTORY не найден."
    exit 1
fi

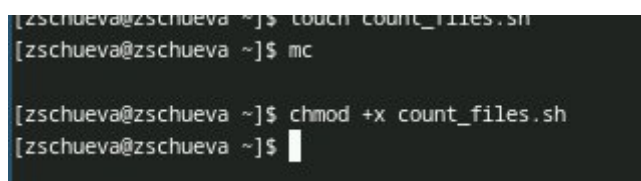
# Считаем количество файлов с заданным расширением
COUNT=$(find "$DIRECTORY" -type f -name ".*$EXTENSION" | wc -l)

echo "Количество файлов с расширением .$EXTENSION в каталоге $DIRECTORY: $COUNT"

```

Рис 4.1

4.2. Сделать скрипт исполняемым командой `chmod +x count_files.sh`



```

[zschueva@zschueva ~]$ touch count_files.sh
[zschueva@zschueva ~]$ mc

[zschueva@zschueva ~]$ chmod +x count_files.sh
[zschueva@zschueva ~]$

```

Рис 4.2

## Выводы

Изучила основы программирования в оболочке ОС UNIX/Linux. Научилась писать небольшие командные файлы.

## Контрольные вопросы

### 1. Понятие командной оболочки

Командная оболочка (или shell) — это программа, которая предоставляет интерфейс для взаимодействия пользователя с операционной системой. Она позволяет пользователям вводить команды, которые затем интерпретируются и выполняются.

### 2. Что такое POSIX?

POSIX (Portable Operating System Interface) — это набор стандартов, определяющих интерфейсы операционных систем, включая командные оболочки и утилиты. POSIX обеспечивает совместимость между различными UNIX-подобными системами, позволяя разработчикам писать переносимые программы.

### 4. Назначение операторов let и read

- **let**: используется для выполнения арифметических операций. Например:

```
```bash
```

```
let "result = a + b"
```

- **read**: используется для чтения ввода от пользователя. Например:

```
```bash
```

```
read -p "Введите ваше имя: " name
```

### 5. Арифметические операции в bash

В bash можно использовать следующие арифметические операции:

- Сложение: ``+``

- Вычитание: ``-``

- Умножение: ``*``

- Деление: ``/``

- Остаток от деления: ``%``

Пример использования:

```
```bash
```

```
result=$((a + b))
```

## 6. Операция (( ))

Операция `(( ))` используется для выполнения арифметических операций в `bash`. Она позволяет выполнять вычисления без необходимости использовать `let` или `$(())`.

Пример:

```
```bash
```

```
(( result = a + b ))
```

## 7. Стандартные имена переменных

Некоторые стандартные имена переменных в `bash`:

- `$HOME`: домашний каталог пользователя.
- `$USER`: имя текущего пользователя.
- `$PWD`: текущий рабочий каталог.
- `$PATH`: список директорий для поиска исполняемых файлов

## 8. Метасимволы

Метасимволы — это специальные символы в командной строке, которые имеют особое значение. Например:

- `*`: соответствует любому числу символов.
- `?`: соответствует одному любому символу.
- `[]`: соответствует любому символу из указанных в скобках

## 9. Экранирование метасимволов

Метасимволы можно экранировать с помощью обратного слэша (`\`). Например, чтобы использовать звездочку как обычный символ, нужно написать `\*`.

Также можно использовать одинарные (`'`) или двойные (`"`) кавычки для экранирования метасимволов.

## 10. Создание и запуск командных файлов



Чтобы создать командный файл:

1. Напишите команды в текстовом редакторе.
2. Сохраните файл с расширением `.sh``.
3. Сделайте файл исполняемым:

## 12. Определение типа файла

Чтобы выяснить, является ли файл каталогом или обычным файлом, можно использовать условие `-d`` для каталогов и `-f`` для обычных файлов:

## 13. Назначение команд `set`, `typeset` и `unset`

- **`**set**`**: используется для установки параметров оболочки и управления переменными.
- **`**typeset**`**: используется для определения переменных и их типов (например, массивов).
- **`**unset**`**: используется для удаления переменной или функции

## 4. Передача параметров в командные файлы

Параметры передаются в командные файлы через позиционные параметры, которые обозначаются как `$1``, `$2``, ..., `$N`` для первого, второго и т.д. параметра. `@$`` и `*$`` используются для доступа ко всем параметрам.

## 15. Специальные переменные языка `bash`

Некоторые специальные переменные в `bash`:

- `$0``: имя скрипта или программы.
- `$1``, `$2``, ..., `$N``: позиционные параметры.
- `$#``: количество переданных параметров.
- `$?``: код завершения последней выполненной команды.
- `$$``: PID текущего процесса.

## Список литературы