

# Лабораторная работа №13

Программирование в командном процессоре ОС UNIX. Ветвления и циклы

---

Полякова Ю.А.

28 февраля 2007

Российский университет дружбы народов, Москва, Россия

Преподаватель Кулябов Д. С., д.ф.-м.н., профессор

## Информация

---

- Полякова Юлия Александровна
- Студент
- Российский университет дружбы народов
- yulya.polyakova.07@mail.ru
- <https://github.com/JuliaMaffin123>



## Вводная часть

---

- Умение работать с командными файлами важно для понимания системы и будет полезно в будущем

- командный процессор ОС UNIX. Командные файлы

- Цель: Изучить основы программирования в оболочке ОС UNIX. Научится писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.
- Задачи:
  - Ознакомиться с теоретическим материалом.
  - Написать 4 командных файла по заданию.
  - Ответить на контрольные вопросы.

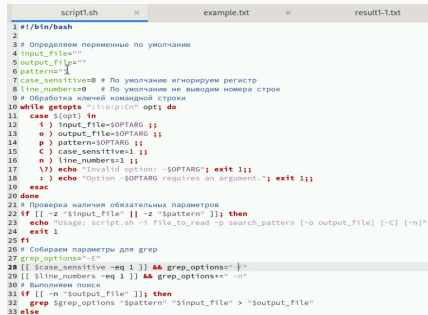
- редактор gedit
- терминал
- командные файлы



## Выполнение лабораторной работы

---

Рассмотрим первый скрипт, который анализирует командную строку с ключами и ищет строки по ключу -p, ключи: -iinputfile — прочитать из указанного файла; -ooutputfile — вывести в указанный файл; -rшаблон — шаблон для поиска; -C — различать большие и малые буквы; -n — выдавать номера строк.

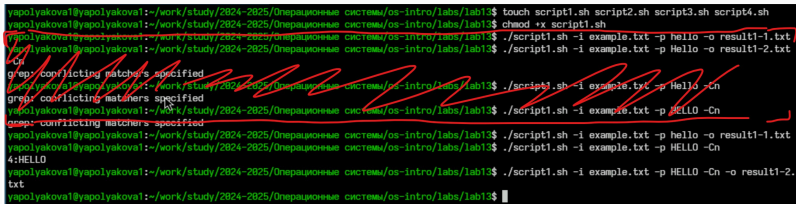


```
1#!/bin/bash
2
3# Определяем переменные по умолчанию
4input_file=""
5output_file=""
6pattern=""
7case_sensitive=0 # По умолчанию игнорируем регистр
8line_numbers=0 # По умолчанию не выводим номера строк
9# Обработка ключей командной строки
10while getopts ":i:o:r:C:n" opt; do
11  case ${opt} in
12    i ) input_file=$OPTARG ;;
13    o ) output_file=$OPTARG ;;
14    p ) pattern=$OPTARG ;;
15    C ) case_sensitive=1 ;;
16    n ) line_numbers=1 ;;
17    \?) echo "Invalid option: -$OPTARG" && exit 1;;
18    : ) echo "Option -$OPTARG requires an argument." && exit 1;;
19  esac
20done
21# Проверка наличия обязательных параметров
22if [[ -z "$input_file" ]] || -z "$pattern" ]; then
23  echo "Usage: script.sh -i file_to_read -p search_pattern [-o output_file] [-C] [-n]"
24  exit 1
25fi
26# Собираем параметры для grep
27grep_options=""
28[[ $case_sensitive -eq 1 ]] && grep_options+=" -C"
29[[ $line_numbers -eq 1 ]] && grep_options+=" -n"
30# Выполняем поиск
31if [[ -n "$output_file" ]]; then
32  grep $grep_options "$pattern" "$input_file" > "$output_file"
33else
```

Рис. 1: Листинг script1.sh

## Запуск script1.sh

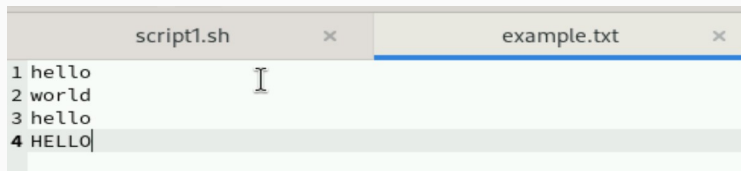
Открываем доступ к файлу и запускаем. В начале мы запустили поиск “hello” в файле “example.txt” без учета регистра и номеров строк в файл result1-1.txt. А затем проверили учет больших букв и номера строк и вывели результат сначала в терминал, а потом в result1-2.txt



```
yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ touch script1.sh script2.sh script3.sh script4.sh
yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ chmod +x script1.sh
yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ ./script1.sh -i example.txt -p hello -o result1-1.txt
yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ ./script1.sh -i example.txt -p Hello -o result1-2.txt
Cn
grep: conflicting matchers specified
yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ ./script1.sh -i example.txt -p Hello -Cn
grep: conflicting matchers specified
yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ ./script1.sh -i example.txt -p HELLO -Cn
grep: conflicting matchers specified
yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ ./script1.sh -i example.txt -p hello -o result1-1.txt
yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ ./script1.sh -i example.txt -p HELLO -Cn
4:HELLO
yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ ./script1.sh -i example.txt -p HELLO -Cn -o result1-2.
txt
yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ █
```

Рис. 2: Запуск script1.sh

Файл example.txt



```
1 hello
2 world
3 hello
4 HELLO
```

Рис. 3: example.txt

Файл result1-1.txt

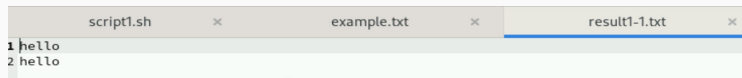


Рис. 4: result1-1.txt

Файл result1-2.txt

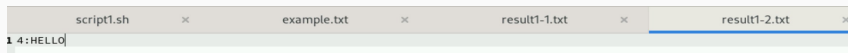


Рис. 5: result1-2.txt

## Листинг check\_number.c

Рассмотрим второе задание. Сначала надо было “Написать на языке Си программу, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю. Затем программа завершается с помощью функции `exit(n)`, передавая информацию в о коде завершения в оболочку.” На фото листинг

```
check_number.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5     int number;
6
7     printf("Введите число: ");
8     scanf("%d", &number);
9
10    if (number > 0)
11        exit(1); // Число больше нуля
12    else if (number < 0)
13        exit(2); // Число меньше нуля
14    else
15        exit(0); // Число равно нулю
16
```

Рис. 6: Листинг check\_number.c

Теперь рассмотрим сам командный файл. Задание: Командный файл должен вызывать эту программу и, проанализировав с помощью команды \$?, выдать сообщение о том, какое число было введено.

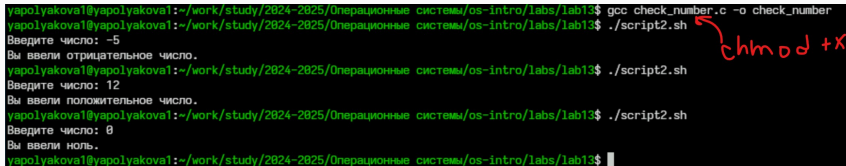
```
script2.sh
1 #!/bin/bash
2
3 ./check_number
4
5 # Анализируем статус выхода
6 case $? in
7     1)
8         echo "Вы ввели положительное число."
9         ;;
10    2)
11        echo "Вы ввели отрицательное число."
12        ;;
13    0)
14        echo "Вы ввели ноль."
15        ;;
16    *)
17        echo "Ошибка!"
18        ;;
19 esac
```

Рис. 7: Листинг script2.sh



Компилируем С-файл с помощью gcc, открываем доступ к командному файлу и запускаем.

Тестируем с разными числами



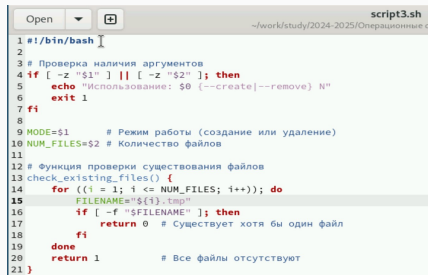
```
yaopolyakova1@yaopolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ gcc check_number.c -o check_number
yaopolyakova1@yaopolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ ./script2.sh
Введите число: -5
Вы ввели отрицательное число.
yaopolyakova1@yaopolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ ./script2.sh
Введите число: 12
Вы ввели положительное число.
yaopolyakova1@yaopolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ ./script2.sh
Введите число: 0
Вы ввели ноль.
yaopolyakova1@yaopolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$
```

A red handwritten note "chmod +x" with an arrow points to the `./script2.sh` command in the terminal output.

Рис. 8: Запуск script2.sh

## Листинг script3.sh - часть 1

Рассмотрим третий скрипт. Задание: Написать командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до N (например 1.tmp, 2.tmp, 3.tmp и т.д.). Число файлов, которые необходимо создать, передаётся в аргументы командной строки. Этот же командный файл должен уметь удалять все созданные им файлы (если они существуют).



```
#!/bin/bash
#
# Проверка наличия аргументов
if [ -z "$1" ] || [ -z "$2" ]; then
    echo "Использование: $0 [--create|--remove] N"
    exit 1
fi
#
MODE=$1 # Режим работы (создание или удаление)
NUM_FILES=$2 # Количество файлов
#
# Функция проверки существования файлов
check_existing_files() {
    for ((i = 1; i <= NUM_FILES; i++)); do
        FILENAME="${i}.tmp"
        if [ -f "$FILENAME" ]; then
            return 0 # Существует хотя бы один файл
        fi
    done
    return 1 # Все файлы отсутствуют
}
```

Рис. 9: Листинг script3.sh - часть 1 (проверка аргум. и сущ-я файлов)

### Основная часть кода script3.sh, работа с режимами

```
23 # Основная логика
24 if [ "$MODE" = "--create" ]; then
25     # Проверим, существуют ли уже файлы
26     if check_existing_files; then
27         echo "Ошибка: Некоторые файлы уже существуют. Остановлено создание."
28         exit 1
29     fi
30
31     # Цикл создания файлов
32     for ((i = 1; i <= NUM_FILES; i++)); do
33         touch "${i}.tmp"
34         echo "Создан файл ${i}.tmp"
35     done
36 elif [ "$MODE" = "--remove" ]; then
37     # Цикл удаления файлов
38     for ((i = 1; i <= NUM_FILES; i++)); do
39         FILENAME="${i}.tmp"
40         if [ -f "$FILENAME" ]; then
41             rm "$FILENAME"
42             echo "Файл $FILENAME удалён."
43         else
44             echo "Файл $FILENAME не существует."
45         fi
46     done
47 else
48     echo "Неверный режим работы. Используйте '--create' или '--remove'"
49     exit 1
50 fi
```

Рис. 10: Листинг script3.sh - часть 2 (основная часть с режимами)

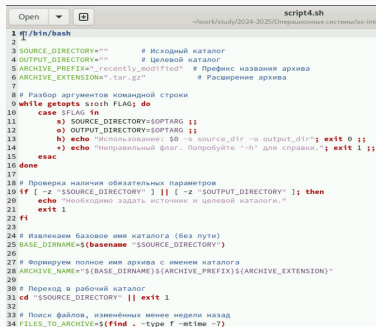
Открываем доступ к исполнению, тестируем на создании и удалении 3-х файлов

```
ya Polyakova1@ya Polyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ chmod +x script3.sh
ya Polyakova1@ya Polyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ ./script3.sh --create 3
Создан файл 1.tmp
Создан файл 2.tmp
Создан файл 3.tmp
ya Polyakova1@ya Polyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ ./script3.sh --remove 3
Файл 1.tmp удалён.
Файл 2.tmp удалён.
Файл 3.tmp удалён.
ya Polyakova1@ya Polyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$
```

Рис. 11: Запуск script3.sh

## Листинг script4.sh - часть 1

Рассмотрим 4-ый командный файл. Задание: Файл, который командой tar запаковывает в архив все файлы в указанной директории. Модифицировать, чтобы запаковывались только те файлы, которые были изменены менее недели назад (использовать find).



```
1 #!/bin/bash
2
3 SOURCE_DIRECTORY=""      # Исходный каталог
4 OUTPUT_DIRECTORY=""      # Целевой каталог
5 ARCHIVE_PREFIX="recently_modified" # Префикс названия архива
6 ARCHIVE_EXTENSION=".tar.gz" # Расширение архива
7
8 # Разбор аргументов командной строки
9 while getopts siozh FLAG; do
10     case $FLAG in
11         s) SOURCE_DIRECTORY=$OPTARG ;;
12         o) OUTPUT_DIRECTORY=$OPTARG ;;
13         h) echo "Использование: $0 -s source_dir -o output_dir"; exit 0 ;;
14         *) echo "Неправильный флаг. Попробуйте '-h' для справки."; exit 1 ;;
15     esac
16 done
17
18 # Проверка наличия обязательных параметров
19 if [ -z "$SOURCE_DIRECTORY" ] || [ -z "$OUTPUT_DIRECTORY" ]; then
20     echo "Необходимо задать источник и целевой каталог."
21     exit 1
22 fi
23
24 # Извлекаем базовое имя каталога (без пути)
25 BASE_DIRNAME=$(basename "$SOURCE_DIRECTORY")
26
27 # Формируем полное имя архива с именем каталога
28 ARCHIVE_NAME="${BASE_DIRNAME}${ARCHIVE_PREFIX}${ARCHIVE_EXTENSION}"
29
30 # Переход в рабочий каталог
31 cd "$SOURCE_DIRECTORY" || exit 1
32
33 # Поиск файлов, изменённых менее недели назад
34 FILES_TO_ARCHIVE=$(find . -type f -mtime -7)
```

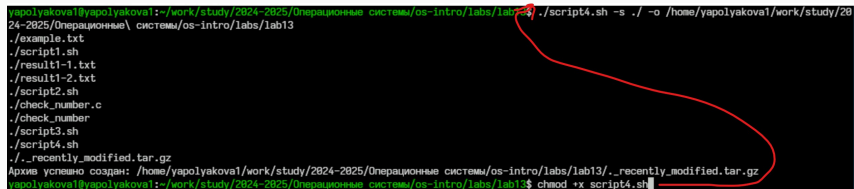
Рис. 12: Листинг script4.sh - часть 1

Конец скрипта с архивацией

```
35
36 # Проверка наличия свежих файлов
37 if [ -z "$FILES_TO_ARCHIVE" ]; then
38     echo "Нет файлов, изменённых менее недели назад."
39     exit 0
40 fi
41
42 # Создание архива
43 tar czvf "$OUTPUT_DIRECTORY/$ARCHIVE_NAME" $FILES_TO_ARCHIVE
44
45 echo "Архив успешно создан: $OUTPUT_DIRECTORY/$ARCHIVE_NAME"
```

Рис. 13: Листинг script4.sh - часть 2

Открываем доступ к исполнению, запускаем, архивируем файлы этой лабораторной работы



```
yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ ./script4.sh -s ./ -o /home/yapolyakova1/work/study/2024-2025/Операционные системы/os-intro/labs/lab13
./example.txt
./script1.sh
./result1-1.txt
./result1-2.txt
./script2.sh
./check_number.c
./check_number
./script3.sh
./script4.sh
./._recently_modified.tar.gz
Архив успешно создан: /home/yapolyakova1/work/study/2024-2025/Операционные системы/os-intro/labs/lab13/._recently_modified.tar.gz
yapolyakova1@yapolyakova1:~/work/study/2024-2025/Операционные системы/os-intro/labs/lab13$ chmod +x script4.sh
```

Рис. 14: Запуск script4.sh

Архив действительно создан

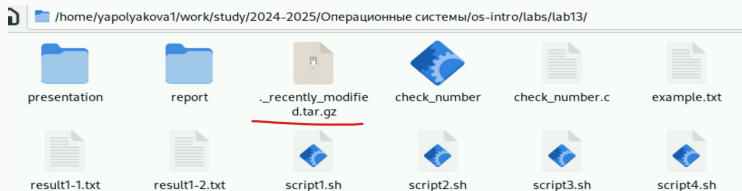


Рис. 15: Архив



## Контрольные вопросы

---

### 1. Каково предназначение команды `getopts`?

Команда **`getopts`** предназначена для удобной обработки аргументов командной строки в сценариях оболочки (shell scripts). Она помогает разбивать переданные аргументы на отдельные элементы и проверять правильность передачи флагов и значений. Обычно применяется совместно с циклом **`while`**, что позволяет автоматизировать обработку сложных комбинаций параметров, обеспечивая безопасность и ясность синтаксиса.

Пример использования:

```
while getopts ab:c OPT; do
    case $OPT in
        a) echo "Опция А была указана";;
        b) echo "Опция В имеет значение: $OPTARG";;
        c) echo "Опция С указана";;
        ?) echo "Неправильно использован флаг";;
    esac
done
```

### 2. Какое отношение метасимволы имеют к генерации имён файлов?

Метасимволы (такие как звездочка `*`, знак вопроса `?`, квадратные скобки `[]`) играют важную роль в построении масок для поиска и подбора файлов в операционной системе. Они позволяют удобно находить группы файлов по общим признакам.

Примеры:

- Маска `*.txt` выберет все файлы с расширением `.txt`.
- Маска `file?.dat` подберёт файлы с названием длиной в четыре символа, начинающиеся с `file` и оканчивающиеся на `.dat`.

Эти маски часто применяются в командах типа `ls`, `rm`, `cp`, что значительно ускоряет выбор файлов.

### 3. Какие операторы управления действиями вы знаете?

Операторы управления помогают контролировать поток выполнения инструкций в сценарии оболочки. Основные операторы включают:

- **if** — оператор условного ветвления, позволяющий исполнять блок кода только при выполнении какого-либо условия.
- **case** — оператор множественного выбора, похожий на switch-case в языках программирования.
- **for, while, until** — операторы организации циклов.
- **break, continue** — операторы изменения хода выполнения цикла.
- **&&** (логическое AND) и **||** (логическое OR) — используются для объединения нескольких условий или команд.

4. Какие операторы используются для прерывания цикла?

Для остановки цикла используются два оператора:

- **break** — немедленно выходит из ближайшего охватывающего цикла (**for**, **while**, **until**).
- **continue** — пропускает оставшуюся часть итерации и переходит сразу к следующей итерации цикла.

Пример использования операторов:

```
for i in $(seq 1 10); do
    if [ $i -gt 5 ]; then break; fi
    echo $i
done
```

5. Для чего нужны команды `false` и `true`?

Командами `false` и `true` являются стандартные команды оболочки, используемые для простых тестов условий. Их назначение следующее:

- **`false`** — всегда возвращает ненулевой код завершения (обычно 1), обозначая ошибку или ложное условие.
- **`true`** — всегда возвращает нулевой код завершения (0), обозначая успешное завершение или истинное условие.

Они полезны в случаях, когда необходимо организовать простую проверку или задать заведомо известное условие.

Пример использования:

```
if true; then  
    echo "Это всегда правда"  
fi
```



6. Что означает строка `if test -f man$s/i.$s`, встречающаяся в командном файле?

Эта строка представляет собой проверку существования файла с определенным форматом. Рассмотрим подробнее:

- `test -f filename` — проверяет, существует ли файл и является ли он обычным файлом (не каталогом, символьной ссылкой и т.п.).
- `man$s/i.$s` — выражение, генерирующее динамическое имя файла. Здесь:
  - `$s` — вероятно, какая-то переменная, заменяющая расширение или категорию файла.
  - `$i` — возможно, индекс или порядковый номер файла.

Таким образом, данная конструкция проверяет, существует ли конкретный файл с определенной структурой имени.

7. Объясните различия между конструкциями `while` и `until`.

Конструкция **`while`**: цикл выполняется, пока условие истинно. Иначе говоря, цикл выполняется снова и снова, пока условие возвращает успех (нулевой код завершения).

Конструкция **`until`**: цикл выполняется, пока условие ложно. Таким образом, цикл выполняется повторно, пока условие возвращает неудачу (ненулевой код завершения).

Были изучены основы программирования в оболочке ОС UNIX. Мы научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.