

# **Отчёта по лабораторной работе №11**

**НКНбд-02-21**

Акондзо Жордани Лади Гаэл

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задание</b>	<b>5</b>
<b>3</b>	<b>Ход работы</b>	<b>7</b>
<b>4</b>	<b>Выводы</b>	<b>15</b>
<b>5</b>	<b>Контрольные вопросы</b>	<b>16</b>

## Список иллюстраций

3.1	Написание программы 1 в emacs . . . . .	8
3.2	вывод программы 1 . . . . .	9
3.3	Написание программы 2.1 в emacs . . . . .	10
3.4	Написание программы 2.2 в emacs . . . . .	10
3.5	вывод программы 2 . . . . .	11
3.6	Написание программы 3 в emacs . . . . .	12
3.7	вывод программы 3 . . . . .	13
3.8	Написание программы 4 в emacs . . . . .	14

# 1 Цель работы

Цель данной работы — Изучить основы программирования в оболочке ОС UNIX. Научится писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

## 2 Задание

1. Используя команды `getopts` `grep`, написать командный файл, который анализирует командную строку с ключами:

- `-iinputfile` — прочитать данные из указанного файла;
- `-ooutputfile` — вывести данные в указанный файл;
- `-ршаблон` — указать шаблон для поиска;
- `-C` — различать большие и малые буквы;
- `-n` — выдавать номера строк.

а затем ищет в указанном файле нужные строки, определяемые ключом `-р`.

2. 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. Командный файл должен вызывать эту программу и, проанализировав с помощью команды `$?`, выдать сообщение о том, какое число было введено.

3. Написать командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до  $\infty$  (например `1.tmp`, `2.tmp`, `3.tmp`, `4.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).

4. Написать командный файл, который с помощью команды `tag` запаковывает в архив все файлы в указанной директории. Модифицировать его так, чтобы

запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду find).

## 3 Ход работы

1. Используя команды `getopts` `grep`, написать командный файл, который анализирует командную строку с ключами:

- `-iinputfile` — прочитать данные из указанного файла;
- `-ooutputfile` — вывести данные в указанный файл;
- `-ршаблон` — указать шаблон для поиска;
- `-C` — различать большие и малые буквы;
- `-n` — выдавать номера строк.

а затем ищет в указанном файле нужные строки, определяемые ключом `-р`. (рис. 3.1, 3.2)

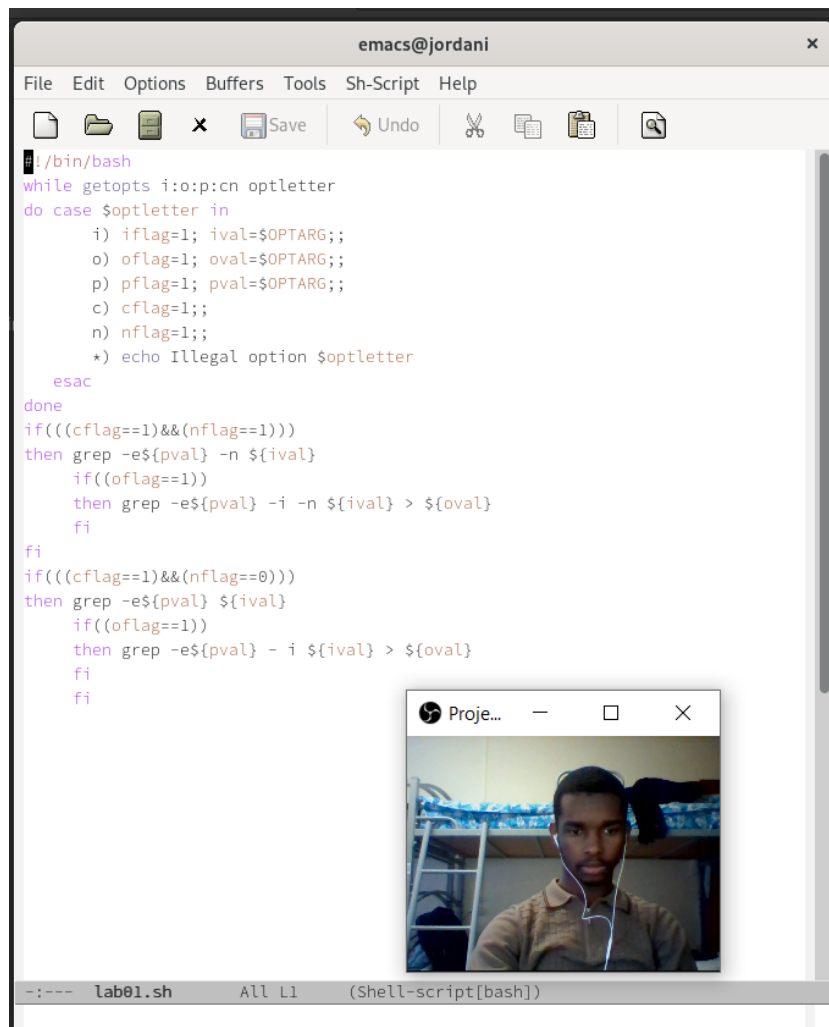


Рис. 3.1: Написание программы 1 в emacs



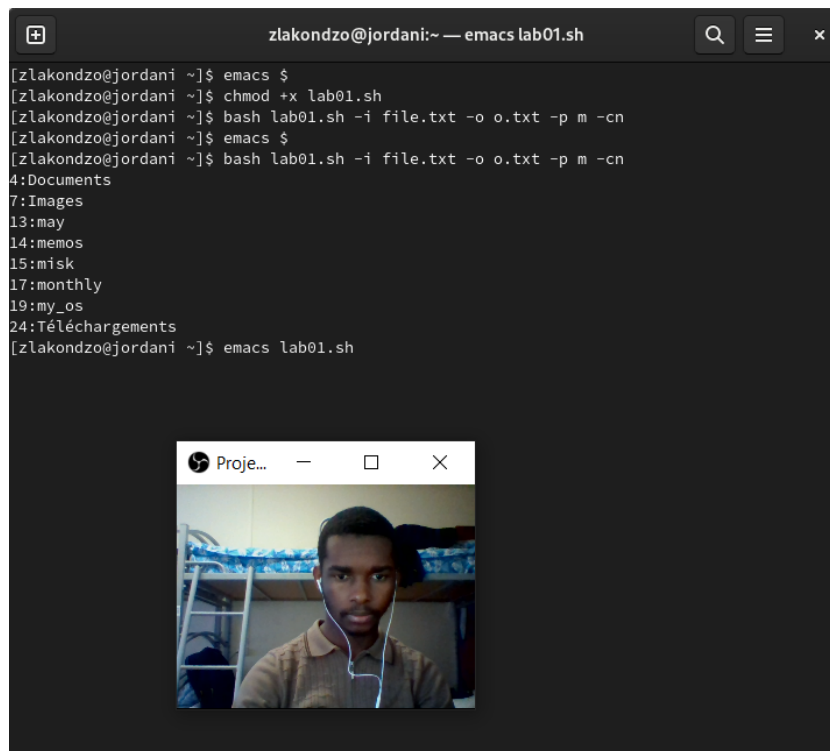


Рис. 3.2: вывод программы 1

2. 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. Командный файл должен вызывать эту программу и, проанализировав с помощью команды `$?`, выдать сообщение о том, какое число было введено. (рис. 3.3, 3.4, 3.5)

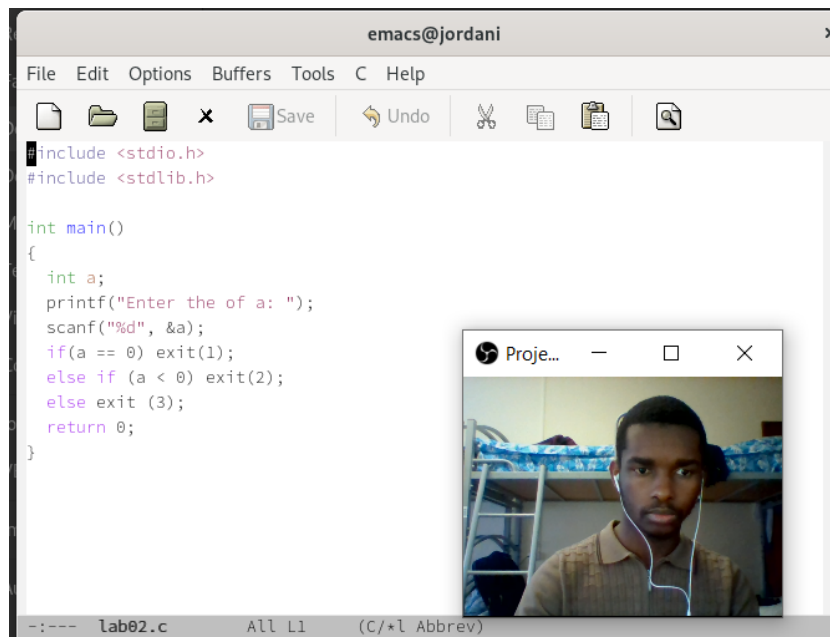


Рис. 3.3: Написание программы 2.1 в emacs

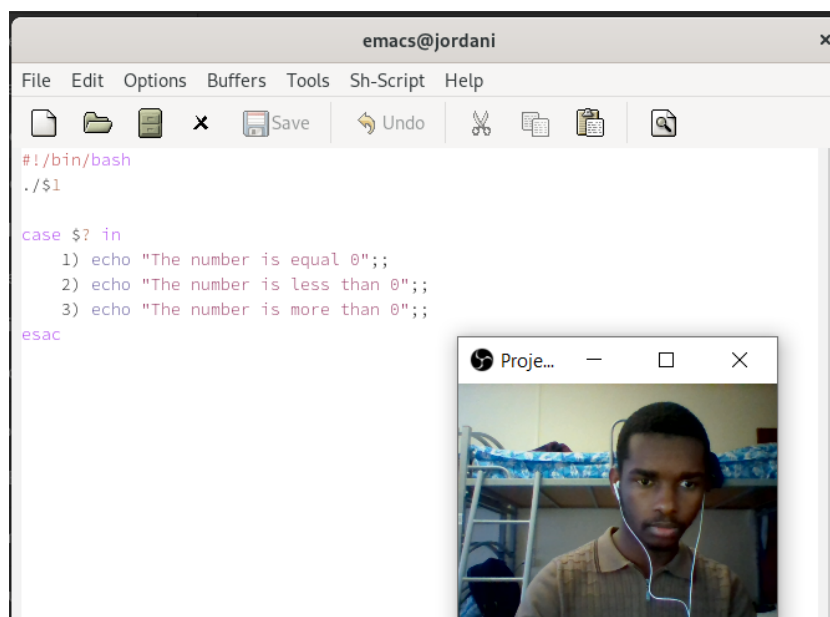
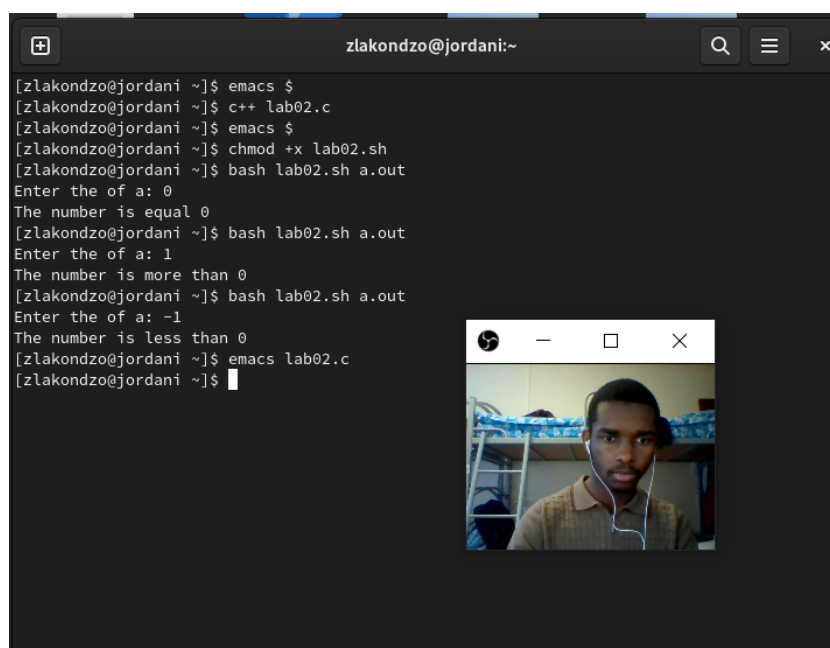


Рис. 3.4: Написание программы 2.2 в emacs



The image shows a terminal window titled 'zlakeondzo@jordani:~'. The terminal contains the following text:

```
[zlakeondzo@jordani ~]$ emacs $
[zlakeondzo@jordani ~]$ c++ lab02.c
[zlakeondzo@jordani ~]$ emacs $
[zlakeondzo@jordani ~]$ chmod +x lab02.sh
[zlakeondzo@jordani ~]$ bash lab02.sh a.out
Enter the of a: 0
The number is equal 0
[zlakeondzo@jordani ~]$ bash lab02.sh a.out
Enter the of a: 1
The number is more than 0
[zlakeondzo@jordani ~]$ bash lab02.sh a.out
Enter the of a: -1
The number is less than 0
[zlakeondzo@jordani ~]$ emacs lab02.c
[zlakeondzo@jordani ~]$
```

Overlaid on the bottom right of the terminal is a small video call window showing a man with dark skin and short hair, wearing a brown shirt and white earbuds. He is looking directly at the camera. The background of the video call shows a room with a blue patterned blanket on a bed and a white ladder.

Рис. 3.5: вывод программы 2

3. Написал командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до  $\infty$  (например 1.tmp, 2.tmp, 3.tmp, 4.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). (рис. 3.6, 3.7)

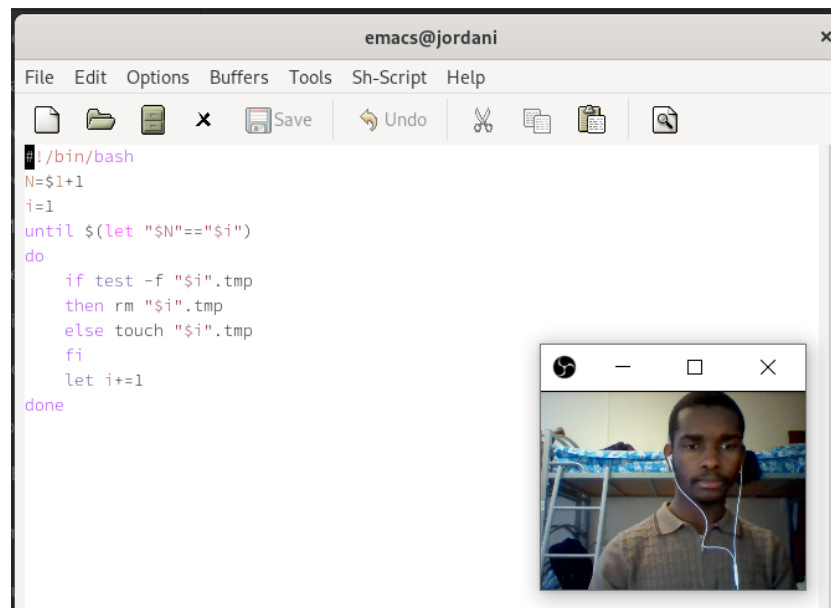


Рис. 3.6: Написание программы 3 в emacs

```
zslakondzo@jordani:~  
[zslakondzo@jordani ~]$ emacs  
[zslakondzo@jordani ~]$ chmod +x lab03.sh  
[zslakondzo@jordani ~]$ bash lab03.sh 6  
[zslakondzo@jordani ~]$ ls  
'$'          Bureau          lab02.c~      monthly      script03.sh  
1.tmp        Documents      lab02.sh      Musique      script04.sh  
2.tmp        file.txt      lab03.sh      my_os        ski.plases  
3.tmp        go            lab03.sh~    oky.sh~     Téléchargements  
4.tmp        Images       lab07.sh      o.txt       text.txt  
5.tmp        install-tl-20220428 lab07.sh~    play        Vidéos  
6.tmp        install-tl-unx.tar.gz letters      Programme_C++ work  
a.out        '<invalid path>' logfile.txt   Public       World  
australia    '<invalid path>.layout' may          reports      Зарпуски  
backup       lab01.sh      memos        Screenshots  
backup.tar.gz lab01.sh~     misk         script01.sh  
bin          lab02.c      Modèles      script02.sh  
[zslakondzo@jordani ~]$ bash lab03.sh 6  
[zslakondzo@jordani ~]$ ls  
'$'          install-tl-20220428 lab07.sh      oky.sh~      ski.plases  
a.out        install-tl-unx.tar.gz lab07.sh~    o.txt       Téléchargements  
australia    '<invalid path>' letters      play        text.txt  
backup       '<invalid path>.layout' logfile.txt   Programme_C++ Vidéos  
backup.tar.gz lab01.sh      may          Public       work  
bin          lab01.sh~    memos        reports      World  
Bureau       lab02.c      misk         Screenshots  Зарпуски  
Documents    lab02.c~    Modèles      script01.sh  
file.txt     lab02.sh    monthly      script02.sh  
go           lab03.sh    Musique      script03.sh  
Images       lab03.sh~   my_os        script04.sh  
[zslakondzo@jordani ~]$
```

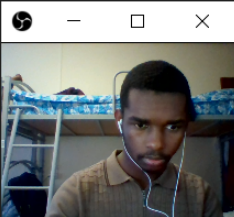


Рис. 3.7: вывод программы 3

4. Написал командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицировать его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду find). (рис. 3.8)

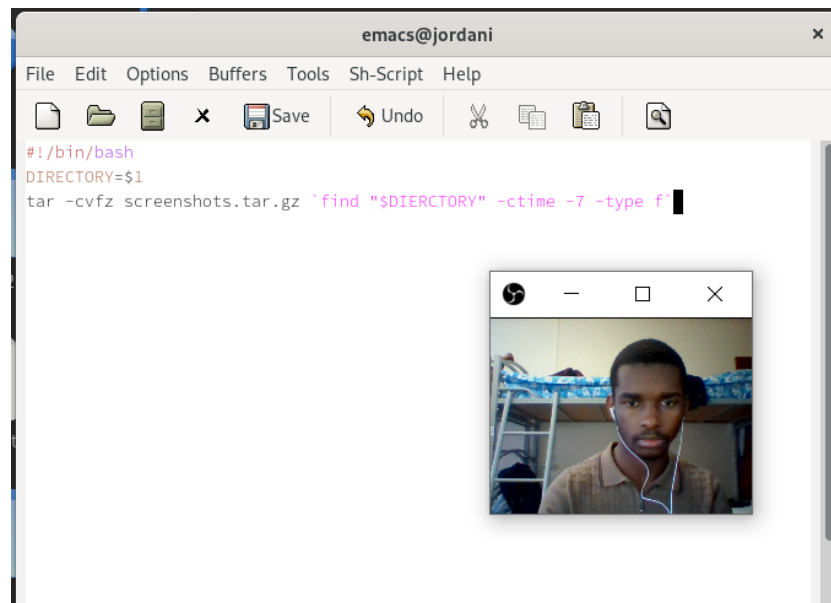


Рис. 3.8: Написание программы 4 в emacs

## 4 Выводы

Во время выполнения работы, мы изучили основы программирования в оболочке ОС UNIX. Научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

## 5 Контрольные вопросы

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

Весьма необходимой при программировании является команда `getopts`, которая осуществляет синтаксический анализ командной строки, выделяя флаги, и используется для объявления переменных. Синтаксис команды следующий: `getopts option-string variable [arg. . . ]`. Флаги – это опции командной строки, обычно помеченные знаком минус; Например, `-F` является флагом для команды `ls -F`. Иногда эти флаги имеют аргументы, связанные с ними. Программы интерпретируют эти флаги, соответствующим образом изменяя свое поведение. Строка опций `option-string` — это список возможных букв и чисел соответствующего флага. Если ожидается, что некоторый флаг будет сопровождаться некоторым аргументом, то за этой буквой должно следовать двоеточие. Соответствующей переменной присваивается буква данной опции. Если команда `getopts` может распознать аргумент, она возвращает истину. Принято включать `getopts` в цикл `while` и анализировать введенные данные с помощью оператора `case`. Предположим, необходимо распознать командную строку следующего формата: `testprog -i file_in.txt -o file_out.doc -L -t -r` Вот как выглядит использование оператора `getopts` в этом случае: 

```
while getopts o:i:Ltr optletter do
case optletter in
o) lag = 1; oval=OPTARG;;
i) iflag=1; ival=$OPTARG;;
L) Lflag=1;;
t) tflag=1;;
r) rflag=1;;
*) echo Illegal option $optletter
esac
done
```

 Функция `getopts` включает две специальные переменные среды – `OPTARG` и `OPTIND`. Если ожидается дополнительное значение, то `OPTARG` устанавливается в значение этого аргумента (будет равна `file_in.txt` для опции `i` и `file_out.doc` для опции `o`). `OPTIND` является числовым индексом на упо-



мянутый аргумент. Функция `getopts` также понимает переменные типа массив, следовательно, можно использовать ее в функции не только для синтаксического анализа аргументов функций, но и для анализа введенных пользователем данных.

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

При перечислении имен 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 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`.

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

Часто бывает необходимо обеспечить проведение каких-либо действий циклически и управление дальнейшими действиями в зависимости от результатов проверки некоторого условия. Для решения подобных задач язык программирования `bash` предоставляет Вам возможность использовать такие управляющие конструкции, как `for`, `case`, `if` и `while`. С точки зрения командного процессора эти управляющие конструкции являются обычными командами и могут использоваться как при создании командных файлов, так и при работе в интерактивном режиме. Команды, реализующие подобные конструкции, по сути дела являются операторами языка программирования `bash`. Поэтому при описании языка программирования `bash` термин оператор будет использоваться наравне с термином команда.

### **4. Какие операторы используются для прерывания цикла?**

Два несложных способа позволяют вам прерывать циклы в оболочке `bash`. Команда `break` завершает выполнение цикла, а команда `continue` завершает данную итерацию блока операторов. Команда `break` полезна для завершения цикла `while` в ситуациях, когда условие перестает быть правильным. Пример бесконечного цикла `while`, с прерыванием в момент, когда файл перестает существовать: `while true do if [! -f $file] then break fi sleep 10 done`

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

Команды ОС UNIX возвращают код завершения, значение которого может быть использовано для принятия решения о дальнейших действиях. Команда `test`, например, создана специально для использования в командных файлах. Единственная функция этой

**6. Что означает строка `if test -f mans/i.$$`, встреченная в командном файле?**

Введенная строка означает условие существования файла `mans/i.$$`

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

Если речь идет о 2-х параллельных действиях, то это `while`. когда мы показываем, что сначала делается 1-е действие. потом оно заканчивается при наступлении 2-го действия, применяем `until`