

Лабораторная работа №12

Дисциплина - операционные системы

Волгин Иван Алексеевич

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	8
4	Выполнение лабораторной работы	12
5	Выводы	18

Список иллюстраций

4.1	Создание файла	12
4.2	Код	13
4.3	Исполнение программы	13
4.4	Создание файла	14
4.5	Содержание каталога	14
4.6	Код	15
4.7	Исполнение программы	15
4.8	Создание файла	16
4.9	Код	16
4.10	Исполнение программы	17

Список таблиц

1 Цель работы

Изучить основы программирования в оболочке ОС UNIX. Научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

2 Задание

1. Написать командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t_1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени $t_2 < t_1$, также выдавая информацию о том, что ресурс используется соответствующим командным файлом (процессом). Запустить командный файл в одном виртуальном терминале в фоновом режиме, перенаправив его вывод в другой (`> /dev/tty#`, где `#` — номер терминала куда перенаправляется вывод), в котором также запущен этот файл, но не фоновом, а в привилегированном режиме. Доработать программу так, чтобы имелась возможность взаимодействия трёх и более процессов.
2. Реализовать команду `man` с помощью командного файла. Изучите содержимое каталога `/usr/share/man/man1`. В нём находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд. Каждый архив можно открыть командой `less` сразу же просмотрев содержимое справки. Командный файл должен получать в виде аргумента командной строки название команды и в виде результата выдавать справку об этой команде или сообщение об отсутствии справки, если соответствующего файла нет в каталоге `man1`.
3. Используя встроенную переменную `$RANDOM`, напишите командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтите, что `$RANDOM` выдаёт псевдослучайные числа в диапазоне от 0 до

32767

3 Теоретическое введение

- Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек: – оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций; – C-оболочка (или csh) — надстройка на оболочке Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд; – оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна; – BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation). POSIX (Portable Operating System Interface for Computer Environments) — 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. Рассмотрим основные элементы программирования в оболочке bash. В других оболочках большинство команд будет совпадать с описанными ниже.

- 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. Целые числа можно записывать как последовательность цифр или в любом базовом формате типа `radix#number`, где `radix` (основание системы счисления) — любое число не более 26. Для большинства команд используются следующие основания систем исчисления: 2 (двоичная), 8 (восьмеричная) и 16 (шестнадцатеричная). Простейшими математическими выражениями являются сложение (+), вычитание (-), умножение (*), целочисленное деление (/) и целочисленный остаток от деления (%). Команда `let` берет два операнда и присваивает их переменной. Положительным моментом команды `let` можно считать то, что для идентификации переменной ей не нужен знак доллара; вы можете писать команды типа `let sum=x+7`, и `let` будет искать переменную `x` и добавлять к ней 7. Команда `let` также расширяет другие выражения `let`, если они заключены в двойные круглые скобки. Таким способом вы можете создавать довольно сложные выражения. Команда `let` не ограничена простыми арифметическими выражениями. Табл. 10.1 показывает полный набор `let`-операций. Подобно `C` оболочка `bash` может присваивать переменной любое значение, а произвольное выражение само имеет значение, которое может использоваться. При этом «ноль» воспринимается как «ложь», а любое другое значение выражения — как «истина». Для облегчения программирования можно записывать условия оболочки `bash` в двойные скобки — (()).

Арифметические операторы оболочки `bash`

Оператор	Синтаксис	Результат
!	<code>!expr</code>	Если <code>expr</code> равно 0, то возвращает 1; иначе 0
!=	<code>expr1 != expr2</code>	Если <code>expr1</code> не равно <code>expr2</code> , то возвращает 1; иначе 0
%	<code>expr1 % expr2</code>	Возвращает остаток от деления <code>expr1</code> на <code>expr2</code>
%=	<code>var=%expr</code>	Присваивает остаток от деления <code>var</code> на <code>expr</code> переменной <code>var</code>
&	<code>expr1 & expr2</code>	Возвращает побитовое AND выражений <code>expr1</code> и <code>expr2</code>

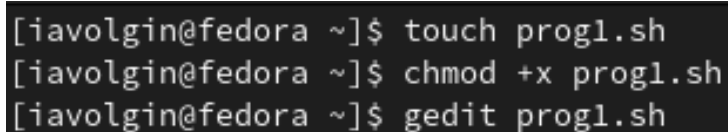
$\text{expr1} \& \text{expr2}$ Если и expr1 и expr2 не равны нулю, то возвращает 1; иначе 0
 $\text{var} \&= \text{expr}$ Присваивает переменной var побитовое AND var и expr
 $\text{var} * \text{expr1} * \text{expr2}$ Умножает expr1 на expr2
 $\text{var} = \text{expr}$ Умножает expr на значение переменной var и присваивает результат переменной
 $\text{var} + \text{expr1} + \text{expr2}$ Складывает expr1 и expr2
 $\text{var} += \text{expr}$ Складывает expr со значением переменной var и результат присваивает переменной
 $\text{var} - \text{expr}$ Операция отрицания expr (унарный минус)
 $\text{expr1} - \text{expr2}$ Вычитает expr2 из expr1
 $\text{var} -= \text{expr}$ Вычитает expr из значения переменной var и присваивает результат переменной
 $\text{var} / \text{expr} / \text{expr2}$ Делит expr1 на expr2
 $\text{var} /= \text{expr}$ Делит значение переменной var на expr и присваивает результат переменной
 $\text{var} < \text{expr1} < \text{expr2}$ Если expr1 меньше, чем expr2 , то возвращает 1, иначе возвращает 0
 $\text{expr1} \ll \text{expr2}$ Сдвигает expr1 влево на expr2 бит
 $\text{var} \ll= \text{expr}$ Побитовый сдвиг влево значения переменной var на expr
 $\text{var} \leq \text{expr1} \leq \text{expr2}$ Если expr1 меньше или равно expr2 , то возвращает 1; иначе возвращает 0
 $\text{var} = \text{expr}$ Присваивает значение expr переменной var
 $\text{var} == \text{expr1} == \text{expr2}$ Если expr1 равно expr2 , то возвращает 1; иначе возвращает 0
 $\text{var} > \text{expr1} > \text{expr2}$ 1, если expr1 больше, чем expr2 ; иначе 0
 $\text{var} \geq \text{expr1} \geq \text{expr2}$ 1, если expr1 больше или равно expr2 ; иначе 0
 $\text{var} \gg \text{expr}$ Сдвигает expr1 вправо на expr2 бит
 $\text{var} \gg= \text{expr}$ Побитовый сдвиг вправо значения переменной var на expr
 $\text{var} ^ \text{expr1} ^ \text{expr2}$ Исключающее OR выражений expr1 и expr2
 $\text{var} ^= \text{expr}$ Присваивает переменной var побитовое XOR var и expr
 $\text{var} | \text{expr1} | \text{expr2}$ Побитовое OR выражений expr1 и expr2
 $\text{var} |= \text{expr}$ Присваивает переменной var результат операции XOR var и expr
 $\text{var} || \text{expr1} || \text{expr2}$ 1, если или expr1 или expr2 являются ненулевыми значениями; иначе 0
 $\text{var} \sim \text{expr}$ Побитовое дополнение до expr

- При перечислении имён файлов текущего каталога можно использовать следующие символы:
 - $*$ — соответствует произвольной, в том числе и пустой строке;
 - $?$ — соответствует любому одинарному символу;
 - $[c1-c2]$ — 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*.

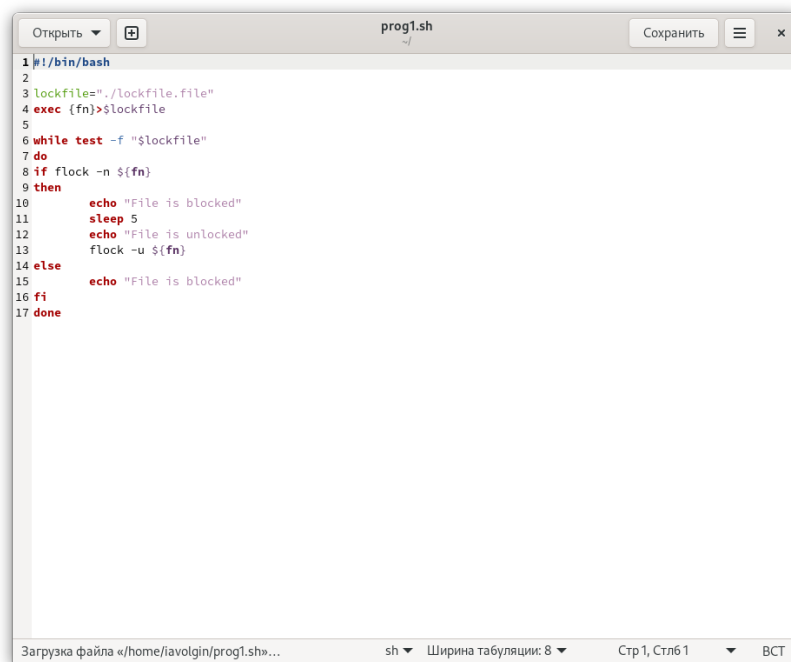
4 Выполнение лабораторной работы

Для выполнения первого задания я создал файл в расширении sh, добавил для него некоторые права (рис. 4.1) и написал в нем код для задания (рис. 4.2). Далее запустил его, чтобы удостовериться в правильности выполнения программы (рис. 4.3).

A terminal window with a dark background and light gray text. It shows three lines of commands being executed by a user named iavolgin on a Fedora system. The first line creates a file named prog1.sh using the 'touch' command. The second line changes the permissions of prog1.sh to be executable (+x) using the 'chmod' command. The third line opens prog1.sh for editing using the 'gedit' command.

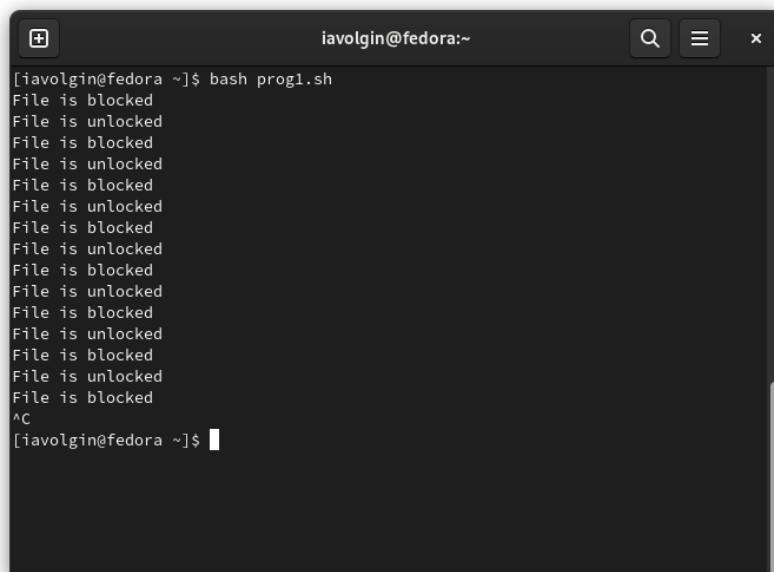
```
[iavolgin@fedora ~]$ touch prog1.sh  
[iavolgin@fedora ~]$ chmod +x prog1.sh  
[iavolgin@fedora ~]$ gedit prog1.sh
```

Рис. 4.1: Создание файла



```
1 #!/bin/bash
2
3 lockfile="/lockfile.file"
4 exec {fn}>$lockfile
5
6 while test -f "$lockfile"
7 do
8   if flock -n ${fn}
9   then
10     echo "File is blocked"
11     sleep 5
12     echo "File is unlocked"
13     flock -u ${fn}
14   else
15     echo "File is blocked"
16   fi
17 done
```

Рис. 4.2: Код



```
iavolgin@fedora:~$ bash prog1.sh
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
File is unlocked
File is blocked
^C
[iavolgin@fedora ~]$
```

Рис. 4.3: Исполнение программы

Далее я приступил к выполнению задания номер 2. Аналогично создал файл и

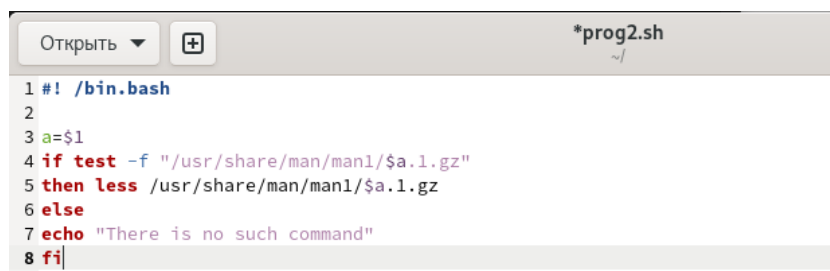
выдал права доступа (рис. 4.4), изучил содержание каталога /usr/share/man/man1 (рис. 4.5), написал код, который выполняет задание (рис. 4.6) и проверил правильность выполнения (рис. 4.7)

```
[iavolgin@fedora ~]$ touch prog2.sh
[iavolgin@fedora ~]$ chmoud +x prog2.sh
bash: chmoud: команда не найдена...
Аналогичная команда: 'chmod'
[iavolgin@fedora ~]$ chmod +x prog2.sh
[iavolgin@fedora ~]$ gedit prog2.sh
```

Рис. 4.4: Создание файла

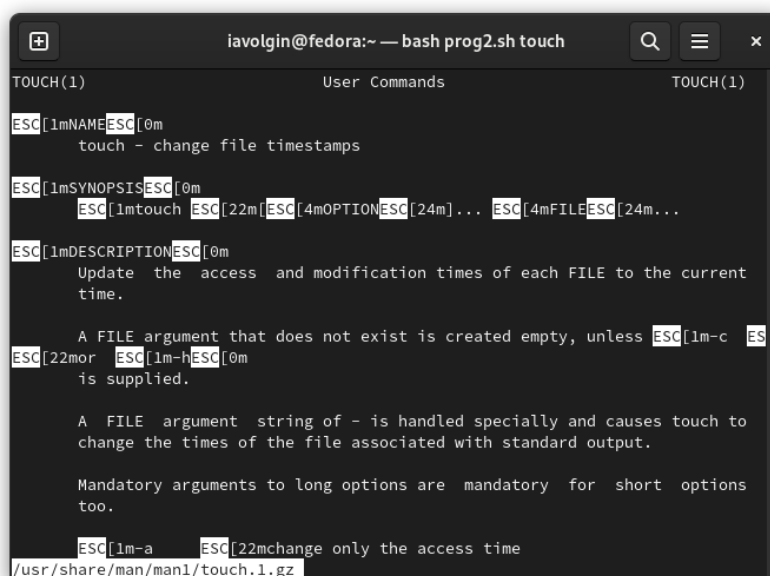
```
yes.1.gz
ypdomainname.1.gz
yum-changelog.1.gz
zcat.1.gz
zcmp.1.gz
zdiff.1.gz
zenity.1.gz
zforce.1.gz
zgrep.1.gz
zip.1.gz
zipcloak.1.gz
zipgrep.1.gz
zipinfo.1.gz
zipnote.1.gz
zipsplit.1.gz
zless.1.gz
zmore.1.gz
znew.1.gz
zsoelim.1.gz
zstd.1.gz
zstdcat.1.gz
zstdgrep.1.gz
zstdless.1.gz
[iavolgin@fedora ~]$ ls /usr/share/man/man1
```

Рис. 4.5: Содержание каталога



```
Открыть + *prog2.sh ~/
1 #!/bin/bash
2
3 a=$1
4 if test -f "/usr/share/man/man1/$a.1.gz"
5 then less /usr/share/man/man1/$a.1.gz
6 else
7 echo "There is no such command"
8 fi
```

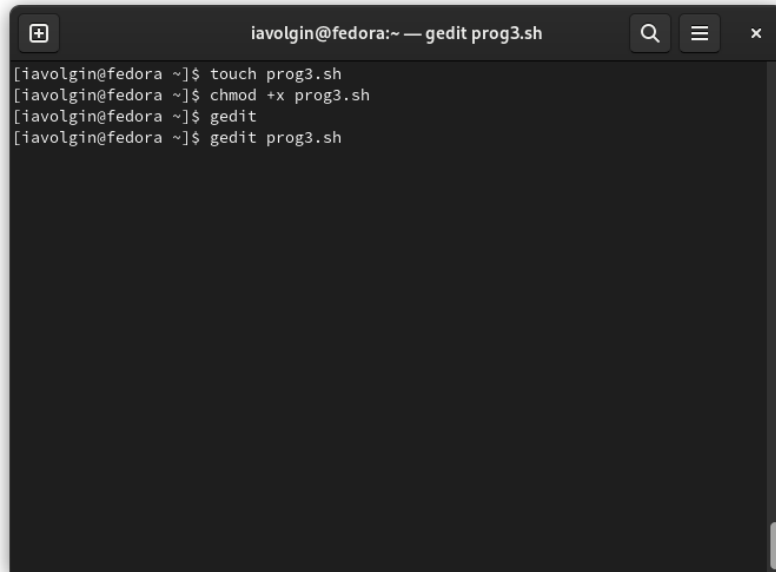
Рис. 4.6: Код



```
iavolgin@fedora:~ — bash prog2.sh touch
TOUCH(1) User Commands TOUCH(1)
ESC[1mNAMEESC[0m
touch - change file timestamps
ESC[1mSYNOPSISESC[0m
ESC[1mtouch ESC[22m[ESC[4mOPTIONESC[24m]... ESC[4mFILEESC[24m]...
ESC[1mDESCRIPTIONESC[0m
Update the access and modification times of each FILE to the current
time.
A FILE argument that does not exist is created empty, unless ESC[1m-c ESC[0m
ESC[22mor ESC[1m-hESC[0m
is supplied.
A FILE argument string of - is handled specially and causes touch to
change the times of the file associated with standard output.
Mandatory arguments to long options are mandatory for short options
too.
ESC[1m-a ESC[22mchange only the access time
/usr/share/man/man1/touch.1.gz
```

Рис. 4.7: Исполнение программы

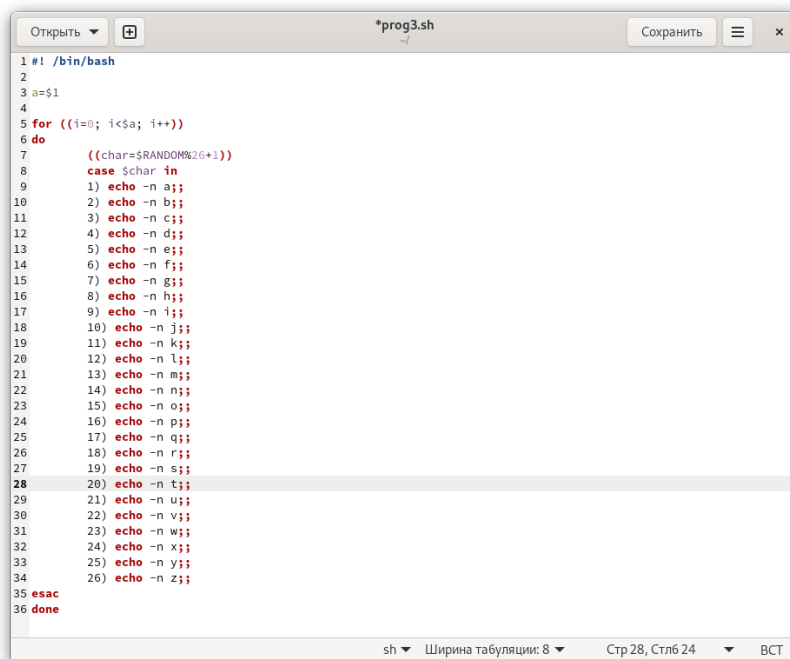
При выполнении третьего задания я аналогично создал файл (рис. 4.7), написал код (рис. 4.8) и далее проверил его, выполнив программу в терминале (рис. 4.9).



A terminal window titled "iavolgin@fedora:~ — gedit prog3.sh". The terminal shows the following commands and their outputs:

```
[iavolgin@fedora ~]$ touch prog3.sh
[iavolgin@fedora ~]$ chmod +x prog3.sh
[iavolgin@fedora ~]$ gedit
[iavolgin@fedora ~]$ gedit prog3.sh
```

Рис. 4.8: Создание файла

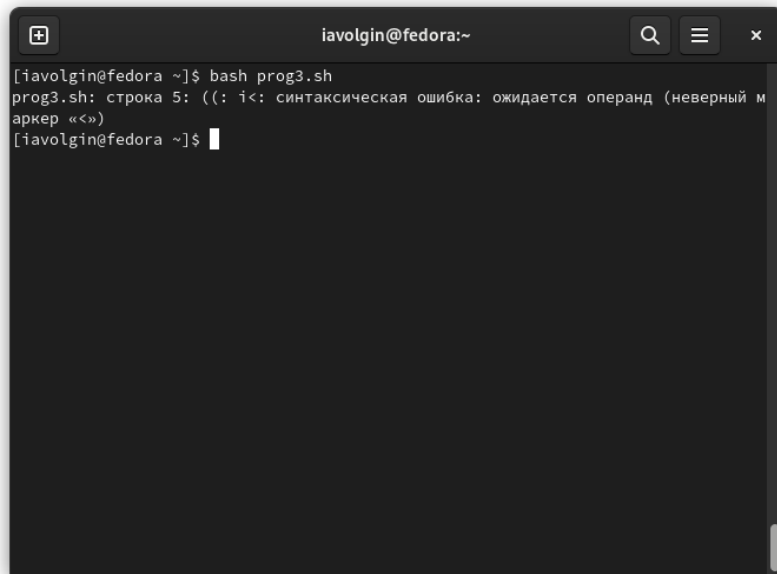


A Gedit editor window titled "*prog3.sh" showing the following code:

```
1 #!/bin/bash
2
3 a=$1
4
5 for ((i=0; i<$a; i++))
6 do
7     ((char=$((RANDOM%26+1)))
8     case $char in
9         1) echo -n a;;
10        2) echo -n b;;
11        3) echo -n c;;
12        4) echo -n d;;
13        5) echo -n e;;
14        6) echo -n f;;
15        7) echo -n g;;
16        8) echo -n h;;
17        9) echo -n i;;
18        10) echo -n j;;
19        11) echo -n k;;
20        12) echo -n l;;
21        13) echo -n m;;
22        14) echo -n n;;
23        15) echo -n o;;
24        16) echo -n p;;
25        17) echo -n q;;
26        18) echo -n r;;
27        19) echo -n s;;
28        20) echo -n t;;
29        21) echo -n u;;
30        22) echo -n v;;
31        23) echo -n w;;
32        24) echo -n x;;
33        25) echo -n y;;
34        26) echo -n z;;
35 esac
36 done
```

The status bar at the bottom shows "sh", "Ширина табуляций: 8", "Стр 28, Стлб 24", and "ВСТ".

Рис. 4.9: Код



A terminal window titled "iavolgin@fedora:~" with standard window controls (maximize, search, menu, close). The terminal shows the execution of a script named "prog3.sh". The prompt is "[iavolgin@fedora ~]\$". The command entered is "bash prog3.sh". The output shows an error message in Russian: "prog3.sh: строка 5: ((: i<: синтаксическая ошибка: ожидается операнд (неверный м аркер «<»)". The prompt returns to "[iavolgin@fedora ~]\$" with a cursor.

```
[iavolgin@fedora ~]$ bash prog3.sh
prog3.sh: строка 5: ((: i<: синтаксическая ошибка: ожидается операнд (неверный м
аркер «<»)
[iavolgin@fedora ~]$
```

Рис. 4.10: Исполнение программы

5 Выводы

В ходе выполнения данной лабораторной работы я Изучил основы программирования в оболочке ОС UNIX, научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.