

Лабораторная работа №12

**Программирование в командном процессоре ОС UNIX. Расширенное
программирование**

Желдакова Виктория Алексеевна

Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение лабораторной работы	8
4	Выводы	12
5	Контрольные вопросы	13

Список иллюстраций

3.1	Первый скрипт	9
3.2	Запуск первого скрипта	9
3.3	Второй скрипт	10
3.4	Запуск второго скрипта	10
3.5	Третий скрипт	11
3.6	Запуск третьего скрипта	11

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 Теоретическое введение

Bash (от англ. Bourne again shell, каламбур «Born again» shell — «возрождённый» shell) — усовершенствованная и модернизированная вариация командной оболочки Bourne shell. Одна из наиболее популярных современных разновидностей командной оболочки UNIX. Особенно популярна в среде Linux, где она часто используется в качестве предустановленной командной оболочки.

Представляет собой командный процессор, работающий, как правило, в интерактивном режиме в текстовом окне. Bash также может читать команды из файла, который называется скриптом (или сценарием). Как и все Unix-оболочки, он поддерживает автодополнение имён файлов и каталогов, подстановку вывода результата команд, переменные, контроль над порядком выполнения, операторы ветвления и цикла. Ключевые слова, синтаксис и другие основные особенности языка были заимствованы из sh. Другие функции, например, история, были скопированы из csh и ksh. Bash в основном соответствует стандарту POSIX, но с рядом расширений.

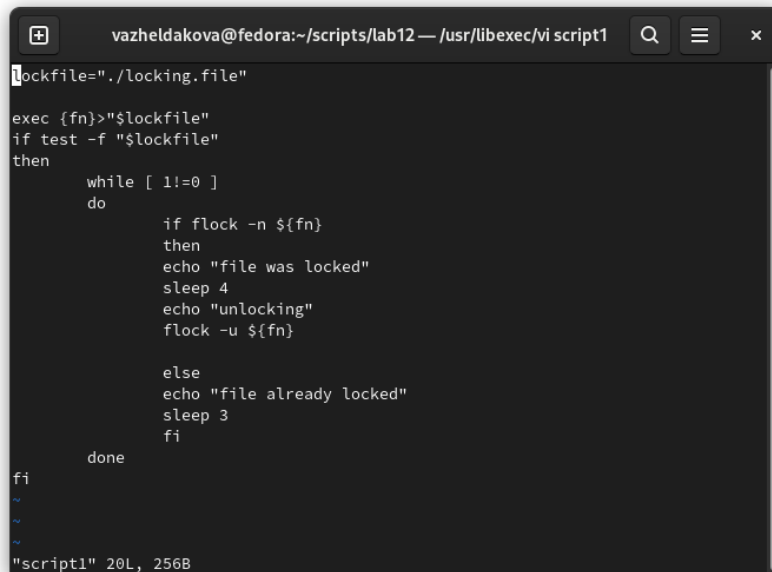
Название «bash» является акронимом от англ. Bourne-again-shell («ещё-одна-командная-оболочка-Борна») и представляет собой игру слов: Bourne-shell — одна из популярных разновидностей командной оболочки для UNIX (sh), автором которой является Стивен Борн (1978), усовершенствована в 1987 году Брайаном Фоксом. Фамилия Bourne (Борн) перекликается с английским словом born, означающим «родившийся», отсюда: рождённая-вновь-командная оболочка.

В сентябре 2014 года в bash была обнаружена широко эксплуатируемая

уязвимость Bashdoor.

3 Выполнение лабораторной работы

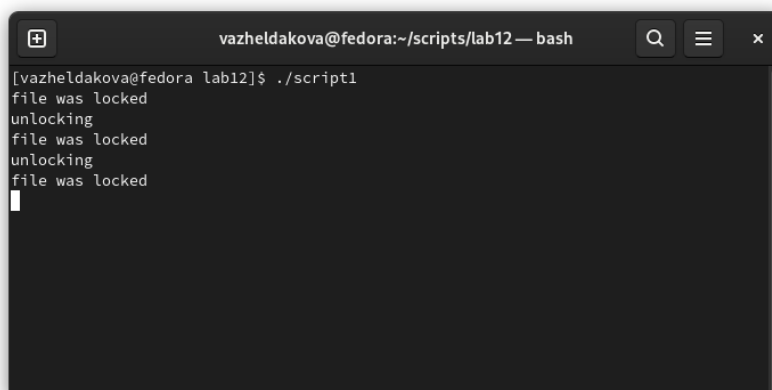
Написали командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t_1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени $t_2 < t_1$, также выдавая информацию о том, что ресурс используется соответствующим командным файлом (процессом). Запустили 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. Доработали программу так, чтобы имелась возможность взаимодействия трёх и более процессов (рис. 3.1 и рис. 3.2).



```
lockfile="./locking.file"
exec {fn}>"$lockfile"
if test -f "$lockfile"
then
    while [ 1!=0 ]
    do
        if flock -n ${fn}
        then
            echo "file was locked"
            sleep 4
            echo "unlocking"
            flock -u ${fn}

        else
            echo "file already locked"
            sleep 3
        fi
    done
fi
"script1" 20L, 256B
```

Рис. 3.1: Первый скрипт




```
[vazheldakova@fedora lab12]$ ./script1
file was locked
unlocking
file was locked
unlocking
file was locked
```

Рис. 3.2: Запуск первого скрипта

Реализовали команду `man` с помощью командного файла. Изучили содержимое каталога `/usr/share/man/man1`. В нем 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 (рис. 3.3 и рис. 3.4).



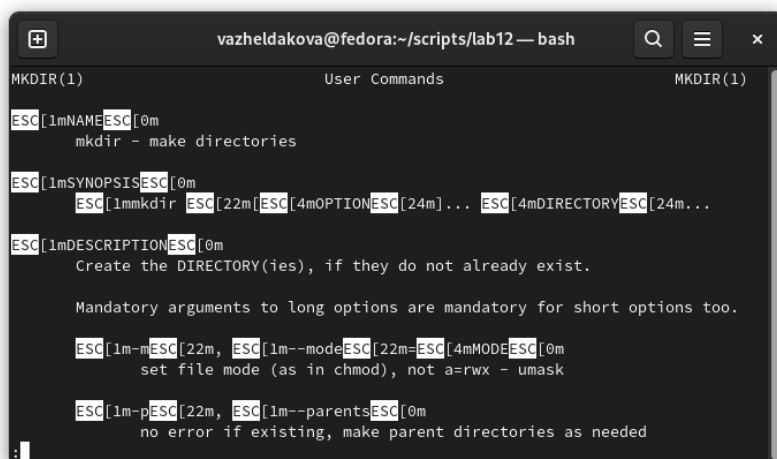
```
vazheldakova@fedora:~/scripts/lab12 — /usr/libexec/vi script2
command=""

while getopts :n: opt
do
case $opt in
n) command="$OPTARG";;
esac
done

if test -f "/usr/share/man/man1/$command.1.gz"
then less /usr/share/man/man1/$command.1.gz
else
echo "no such command"
fi

:
```

Рис. 3.3: Второй скрипт



```
vazheldakova@fedora:~/scripts/lab12 — bash
MKDIR(1)                                User Commands                                MKDIR(1)

ESC[1mNAMEESC[0m
mkdir - make directories

ESC[1mSYNOPSISESC[0m
ESC[1mmkdir ESC[22mESC[4mOPTIONESC[24m]... ESC[4mDIRECTORYESC[24m]...

ESC[1mDESCRIPTIONESC[0m
Create the DIRECTORY(ies), if they do not already exist.

Mandatory arguments to long options are mandatory for short options too.

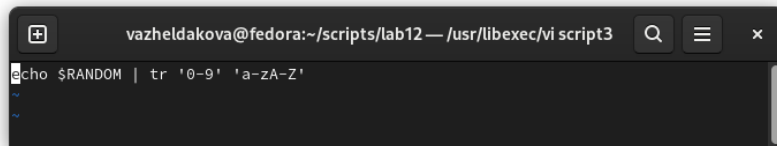
ESC[1m-mESC[22m, ESC[1m--modeESC[22m=ESC[4mMODEESC[0m
set file mode (as in chmod), not a=rwx - umask

ESC[1m-pESC[22m, ESC[1m--parentsESC[0m
no error if existing, make parent directories as needed

:
```

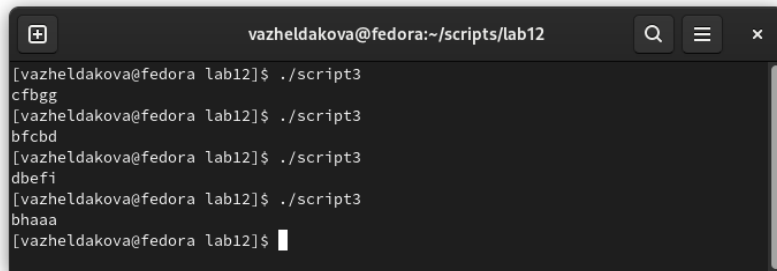
Рис. 3.4: Запуск второго скрипта

Используя встроенную переменную \$RANDOM, написали командный файл, генерирующий случайную последовательность букв латинского алфавита. Учли, что \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767 (рис. 3.5 и рис. 3.6).



```
vazheldakova@fedora:~/scripts/lab12 — /usr/libexec/vi script3
echo $RANDOM | tr '0-9' 'a-zA-Z'
```

Рис. 3.5: Третий скрипт



```
vazheldakova@fedora:~/scripts/lab12
[vazheldakova@fedora lab12]$ ./script3
cfbgg
[vazheldakova@fedora lab12]$ ./script3
bfcdb
[vazheldakova@fedora lab12]$ ./script3
dbefi
[vazheldakova@fedora lab12]$ ./script3
bhaaa
[vazheldakova@fedora lab12]$
```

Рис. 3.6: Запуск третьего скрипта

4 Выводы

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

5 Контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке: `while [$1 != "exit"]`

Между скобками должны быть пробелы, иначе символы в скобках и сами скобки будут восприняты как один элемент.

2. Как объединить (конкатенация) несколько строк в одну?

```
cat file.txt | xargs | sed -e 's/./.\n/g'
```

3. Найдите информацию об утилите `seq`. Какими иными способами можно реализовать её функционал при программировании на `bash`?

Команда `seq` выводит последовательность целых или действительных чисел, подходящую для передачи в другие программы. Реализовать ее функционал можно командой `for n in {1..5} do done`

4. Какой результат даст вычисление выражения `$((10/3))`?

3

5. Укажите кратко основные отличия командной оболочки `zsh` от `bash`.

`Zsh` очень сильно упрощает работу. Но существуют различия. Например, в `zsh` после `for` обязательно вставлять пробел, нумерация массивов в `zsh` начинается с 1. Если вы собираетесь писать скрипт, который будет запускать множество разработчиков, то рекомендуется `Bash`. Если скрипты вам не нужны - `Zsh`.

6. Проверьте, верен ли синтаксис данной конструкции `for ((a=1; a <= LIMIT; a++))`

Верен

7. Сравните язык `bash` с какими-либо языками программирования. Какие преимущества у `bash` по сравнению с ними? Какие недостатки?

Очевидно, что Python более удобен для пользователя и, поскольку Python компилируется байтами, он обычно быстрее. Преимущество Bash это его повсеместное распространение. Также Bash позволяет очень легко работать с файловой системой без лишних конструкций (в отличие от обычного языка программирования). Но относительно обычных языков программирования `bash` очень сжат. Тот же Си имеет гораздо более широкие возможности для разработчика.