

# **Отчёт по лабораторной работе №14**

**Программирование в командном процессоре ОС UNIX.**

Ахмед Нурыев

# Содержание

1	Цель работы	4
2	Выполнение лабораторной работы	5
3	Вывод	7
4	Контрольные вопросы	8

# List of Figures

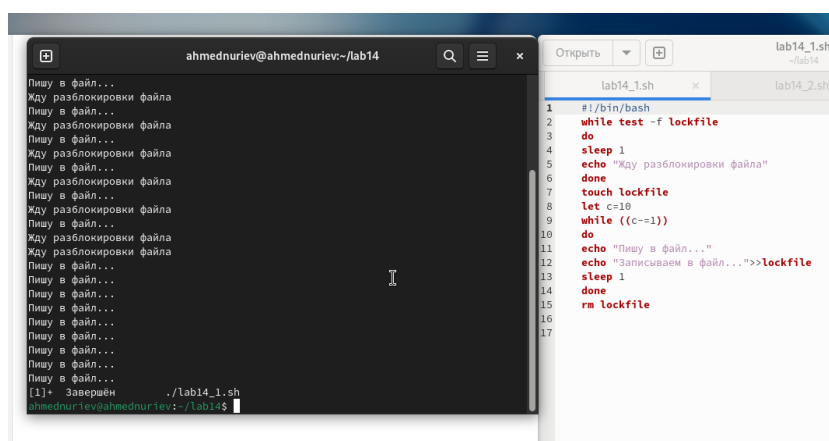
2.1	Задание 1 . . . . .	5
2.2	Задание 2 . . . . .	6
2.3	Задание 3 . . . . .	6

# 1 Цель работы

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

## 2 Выполнение лабораторной работы

1. Написали командный файл, реализующий упрощённый механизм семафоров. Командный файл в течение некоторого времени  $t_1$  дожидается освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использует 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.



The image shows a terminal window on the left and a code editor on the right. The terminal window, titled 'ahmednuriev@ahmednuriev:~/lab14', displays the output of a script: 'Пишу в файл...' (Writing to file...) followed by 'Жду разблокировки файла' (Waiting for file unlock) repeated many times. The code editor, titled 'lab14\_1.sh', shows the script's source code. The script uses a while loop to wait for a lockfile, then a do-while loop to write to the file and sleep for 1 second, and finally removes the lockfile.

```
1 #!/bin/bash
2 while test -f lockfile
3 do
4     sleep 1
5     echo "Жду разблокировки файла"
6 done
7 touch lockfile
8 let c=10
9 while ((c--))
10 do
11     echo "Пишу в файл..."
12     echo "Записываем в файл...">>lockfile
13     sleep 1
14 done
15 rm lockfile
16
17
```

Figure 2.1: Задание 1

2. Реализовали команду `map` с помощью командного файла. Изучили содер-

жимое 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 получает в виде аргумента командной строки название команды и в виде результата выдает справку об этой команде или сообщение об отсутствии справки, если соответствующего файла нет в каталоге `man1`.

```

ahmednuriev@ahmednuriev:~/lab14 — /bin/bash ./lab14_2.sh p...
ESC[4mPWDESC[24m(1)      User Commands
ESC[4mPWDESC[24m(1)
ESC[1mNAMEESC[0m
pwd - print name of current/working directory
ESC[1mSYNOPSISESC[0m
ESC[1mpwdESC[22mESC[4mOPTIONESC[24m...
ESC[1mDESCRIPTIONESC[0m
Print the full filename of the current working directory.
ESC[1m-lESC[22m, ESC[1m--logicalESC[0m
use PWD from environment, even if it contains symlinks
ESC[1m-pESC[22m, ESC[1m--physicalESC[0m
avoid all symlinks
ESC[1m--helpESC[22mdisplay this help and exit
ESC[1m--versionESC[0m
output version information and exit
/usr/share/man/man1/pwd.1.gz
  
```

Figure 2.2: Задание 2

- Используя встроенную переменную `$RANDOM`, написали командный файл, генерирующий случайную последовательность букв латинского алфавита

```

ahmednuriev@ahmednuriev:~/lab14
kydpjsjjl
ahmednuriev@ahmednuriev:~/lab14$ ./lab14_3.sh
wnmdchxnn
ahmednuriev@ahmednuriev:~/lab14$ ./lab14_3.sh
benndqsdK
ahmednuriev@ahmednuriev:~/lab14$ ./lab14_3.sh
appljkyak
ahmednuriev@ahmednuriev:~/lab14$ ./lab14_3.sh
sefitwdt
ahmednuriev@ahmednuriev:~/lab14$ ./lab14_3.sh
udfydssrp
ahmednuriev@ahmednuriev:~/lab14$ ./lab14_3.sh
freejglte
ahmednuriev@ahmednuriev:~/lab14$ ./lab14_3.sh
wdheqneo
ahmednuriev@ahmednuriev:~/lab14$ ./lab14_3.sh
vrlytplpx
ahmednuriev@ahmednuriev:~/lab14$ ./lab14_3.sh
foteitric
ahmednuriev@ahmednuriev:~/lab14$ ./lab14_3.sh
aijscbvpg
ahmednuriev@ahmednuriev:~/lab14$ ./lab14_3.sh
lmojbyrwc
ahmednuriev@ahmednuriev:~/lab14$
  
```

Figure 2.3: Задание 3

## **3 Вывод**

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

## 4 Контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке: `while [$1 != "exit"]` Ответ: Правильный вариант: `while ["$1" != " exit" ]`
2. Как объединить (конкатенация) несколько строк в одну? Ответ: Объединение нескольких строк в одну в Bash происходит с помощью символа `'\ '`
3. Найдите информацию об утилите `seq`. Какими иными способами можно реализовать её функционал при программировании на `bash`? Ответ для примера: В Linux имеется программа `seq`, которая воспринимает в качестве аргументов два числа и выдает последовательность всех чисел, расположенных между заданными. С помощью этой команды можно заставить `for` в `bash` работать точно так же, как аналогичный оператор работает в обычных языках программирования. Для этого достаточно записать цикл `for` следующим образом:

```
for a in $( seq 1 10 ) ; do
catfile_$a
done
```

Эта команда выводит на экран содержимое 10-ти файлов:  
“file\_1”, ..., “file\_10”.

4. Какой результат даст вычисление выражения `$((10/3))`? Ответ: Так как это целочисленное деление, то произойдет округление в сторону ближайшего числа, и выведется 3.  $10/3 = 3$ .



5. Укажите кратко основные отличия командной оболочки zsh от bash. Ответ:

По размеру Bash больше Zsh. Zsh и Bash предлагают сходный функционал. Обе имеют программируемое дополнение (хотя у Zsh оно появилось раньше), встроенные команды и функции для создания скриптов. У Zsh также в запасе есть несколько собственных хитростей, например, расширенная подстановка имени файла, которая превращает команду поиска find почти что в ненужное излишество. Включение в путь \*\* означает соответствие любому символу, включая разделитель - слэш, поэтому \*\*/\*.jpg касается всех файлов \*.jpg в текущей директории и в любых поддиректориях. Мало того, сюда также включаются права доступа к файлу, владелец, тип или отметка времени – большинство опций, предусмотренных find. Например, можно использовать ls -l /\*\*/bin/\*(s) для вывода списка всех setuid-файлов в /bin, /usr/bin и /usr/local/bin. При наборе имени директории в командной строке Zsh переключается на эту директорию. Выполнение скриптов в Zsh основном быстрее, чем в Bash – по большей части примерно на 20% – однако Zsh разработан для интерактивного пользования В Zsh расширенная подстановка имени файла и более развитая опция дополнения..

6. Проверьте, верен ли синтаксис данной конструкции for ((a=1; a <= LIMIT; a++)) Ответ: В bash для оператора цикла for существует другая конструкция.

Пример:

```
for A in Ai Bi Ci do
echo A
done
```

на терминал будет выведено :

```
Ai Bi Ci
```

7. Сравните язык bash с какими-либо языками программирования. Какие

преимущества у `bash` по сравнению с ними? Какие недостатки? Ответ: Вначале был Bourne Shell (`sh`), его написал Стивен Борн для Bell Labs Research Unix. Bash – это Bourne Again Shell (Снова Оболочка Борна), который, к счастью, редко используется. Почти все современные дистрибутивы Linux используют Bash в качестве оболочки по умолчанию, и это превращает Bash в фактический стандарт, с которым сравниваются все остальные. Дело не в малом размере Bash, и не в скорости. По размеру Bash больше некоторых оболочек, кроме одной: Sash, которая не использует библиотек и имеет несколько дополнительных встроенных команд. Bash также и не самая быстрая оболочка, однако большинству пользователей это неважно, ибо подлинно важна его гибкость. Bash обладает некоторыми функциями, превосходящими стандарт POSIX, хотя при желании можно добиться от него и POSIX-поведения. Если запустить Bash командой `sh`, с опцией командной строки `-posix` или при установленной переменной окружения `POSIXLY_CORRECT`, Bash будет работать как стандартная оболочка POSIX. При запуске через `sh`, Bash по возможности пытается работать как исходная оболочка Борна, но лишь в тех ситуациях, когда это не вступит в конфликт со стандартом POSIX.