

# **Отчет по лабораторной работе №14**

**Простейший вариант**

Кривобоков Юрий Дмитриевич

# Содержание

<b>1</b>	<b>Цель</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>8</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>9</b>
<b>5</b>	<b>Контрольные вопросы</b>	<b>12</b>
<b>6</b>	<b>Выводы</b>	<b>15</b>
	<b>Список литературы</b>	<b>16</b>

## Список иллюстраций

4.1	1	.....	9
4.2	2	.....	9
4.3	3	.....	10
4.4	4	.....	10
4.5	5	.....	10
4.6	6	.....	11

## **Список таблиц**

# 1 Цель

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 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`.
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:

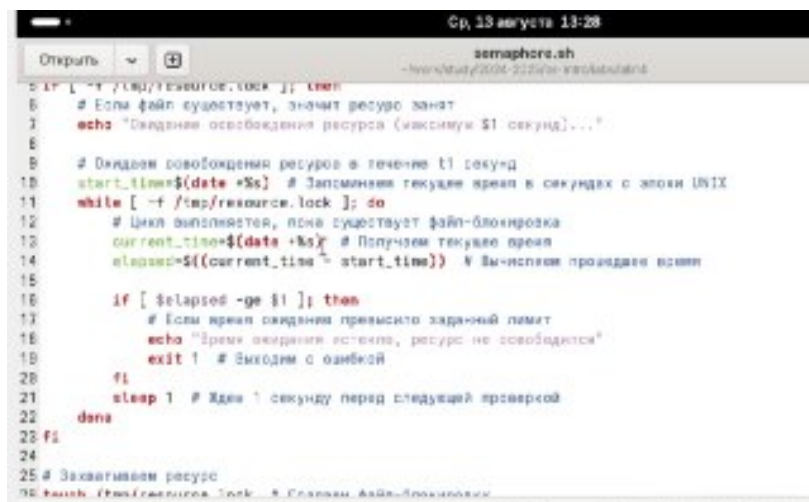
- оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;
- C-оболочка (или csh) — надстройка на оболочкой Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд;
- оболочка Корна (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.



## 4 Выполнение лабораторной работы

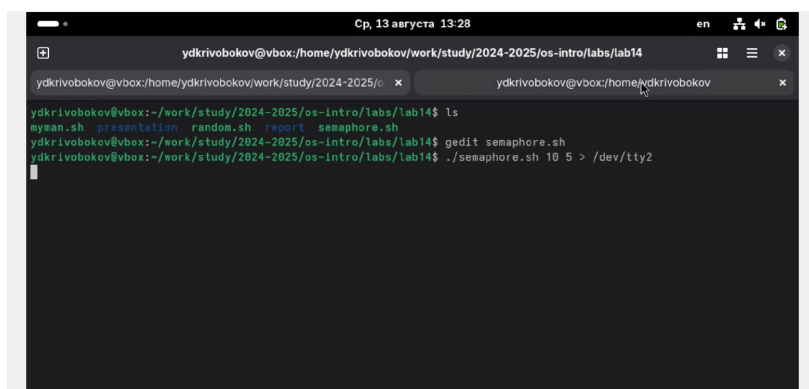
1. Пишу скрипт. (рис. fig:001).



```
1 #!/bin/sh
2 # Если файл существует, значит ресурс занят
3 echo "Ожидание освобождения ресурса (максимум $1 секунд)..."
4
5 # Ожидаем освобождения ресурса в течение t1 секунд
6 start_time=$(date +%s) # Запоминаем текущее время в секундах с эпохи UNIX
7 while [ -f /tmp/resource.lock ]; do
8     # Цикл выполняется, пока существует файл-блокировка
9     current_time=$(date +%s) # Получаем текущее время
10    elapsed=$((current_time - start_time)) # Вычисляем прошедшее время
11
12    if [ $elapsed -ge $1 ]; then
13        # Если время ожидания превысило заданный лимит
14        echo "Время ожидания истекло, ресурс не освобождается"
15        exit 1 # Выходим с ошибкой
16    fi
17    sleep 1 # Ждем 1 секунду перед следующей проверкой
18 done
19 # Заканчиваем ресурс
```

Рисунок 4.1: 1

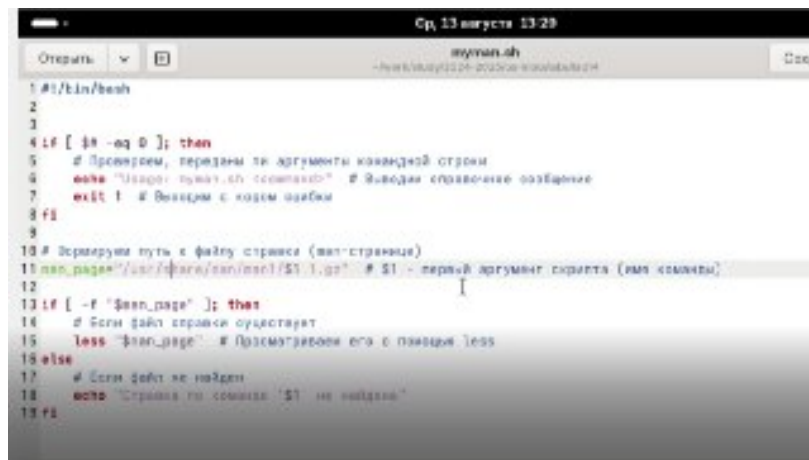
2. Демонстрирую работу. (рис. fig:002).



```
ydkrivobokov@vbox:/home/ydkrivobokov/work/study/2024-2025/os-intro/labs/lab14$ ls
mysan.sh presentation random.sh report semaphore.sh
ydkrivobokov@vbox:/home/ydkrivobokov/work/study/2024-2025/os-intro/labs/lab14$ gedit semaphore.sh
ydkrivobokov@vbox:/home/ydkrivobokov/work/study/2024-2025/os-intro/labs/lab14$ ./semaphore.sh 10 5 > /dev/tty2
```

Рисунок 4.2: 2

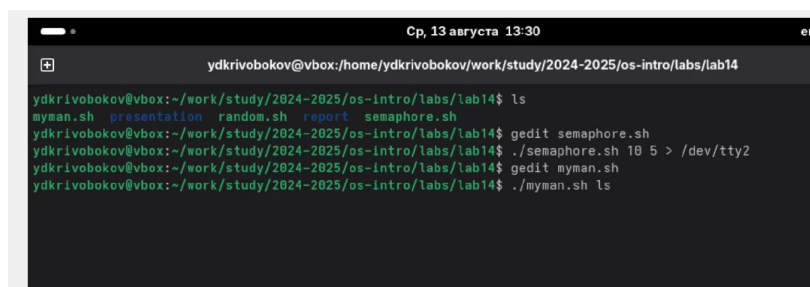
3. Пишу скрипт. (рис. **fig:003**).



```
1 #!/bin/bash
2
3
4 if [ $# -eq 0 ]; then
5     # Проверяем, переданы ли аргументы командной строки
6     echo "Использует myman.sh с параметром" # Выводит справочное сообщение
7     exit 1 # Выходим с кодом ошибки
8 fi
9
10 # Проверяем путь к файлу справки (ман-страница)
11 man_page="/usr/share/man/man1/${1}.gz" # $1 - первый аргумент скрипта (имя команды)
12
13 if [ -f "$man_page" ]; then
14     # Если файл справки существует
15     less "$man_page" # Просматриваем его с помощью less
16 else
17     # Если файл не найден
18     echo "Страница по команде '$1' не найдена"
19 fi
```

Рисунок 4.3: 3

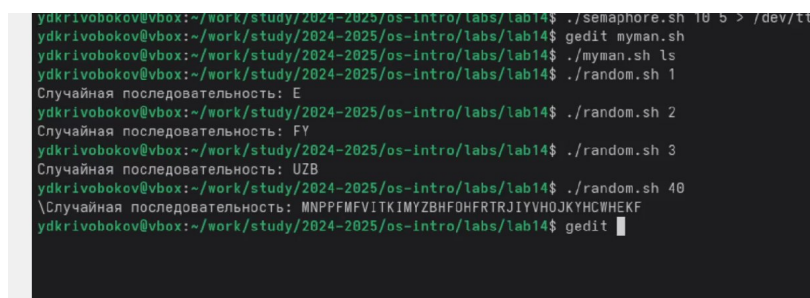
4. Демонстрирую работу. (рис. **fig:004**).



```
ydkrivobokov@vbox:~/work/study/2024-2025/os-intro/labs/lab14$ ls
myman.sh  presentation  random.sh  report  semaphore.sh
ydkrivobokov@vbox:~/work/study/2024-2025/os-intro/labs/lab14$ gedit myman.sh
ydkrivobokov@vbox:~/work/study/2024-2025/os-intro/labs/lab14$ ./semaphore.sh 10 5 > /dev/tty2
ydkrivobokov@vbox:~/work/study/2024-2025/os-intro/labs/lab14$ gedit myman.sh
ydkrivobokov@vbox:~/work/study/2024-2025/os-intro/labs/lab14$ ./myman.sh ls
```

Рисунок 4.4: 4

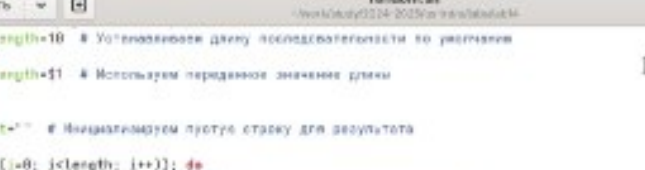
5. Пишу скрипт. (рис. **fig:005**).



```
ydkrivobokov@vbox:~/work/study/2024-2025/os-intro/labs/lab14$ ./semaphore.sh 10 5 > /dev/tty2
ydkrivobokov@vbox:~/work/study/2024-2025/os-intro/labs/lab14$ gedit myman.sh
ydkrivobokov@vbox:~/work/study/2024-2025/os-intro/labs/lab14$ ./myman.sh ls
ydkrivobokov@vbox:~/work/study/2024-2025/os-intro/labs/lab14$ ./random.sh 1
Случайная последовательность: E
ydkrivobokov@vbox:~/work/study/2024-2025/os-intro/labs/lab14$ ./random.sh 2
Случайная последовательность: FY
ydkrivobokov@vbox:~/work/study/2024-2025/os-intro/labs/lab14$ ./random.sh 3
Случайная последовательность: UZB
ydkrivobokov@vbox:~/work/study/2024-2025/os-intro/labs/lab14$ ./random.sh 40
\Случайная последовательность: MNPPFFMVFVITKIMYZBHFQHFRTJJIYVHOJKYHCWHEKF
ydkrivobokov@vbox:~/work/study/2024-2025/os-intro/labs/lab14$ gedit
```

Рисунок 4.5: 5

6. Демонстрирую работу. (рис. **fig:006**).



The screenshot shows a Windows command prompt window with the title "C:\\$ - 13 апреля 2015 13:30". The prompt is "C:\\$". The user has entered "powershell", and the prompt has changed to "PS C:\\$". The user then enters "cd random.sh", and the prompt changes to "PS C:\random.sh". The user enters "type random.ps1", and the contents of the file are displayed:

```

1  length=10 # Устанавливаем длину последовательности по умолчанию
2  # also
3  length=11 # Используем переопределенное значение длины
4  # fi
5
6  result="" # Инициализируем пустую строку для результата
7
8  for ((i=0; i<length; i++)); do
9      # Цикл выполняется length раз
10     # Генерируем случайное число от 0 до 25 (соответствует A-Z)
11     rand=$((RANDOM % 26)) # RANDOM дает число 0-32767, 320 - остаток от деления на 26
12
13     # Преобразуем число в букву (A-Z в ASCII)
14     # printf %c преобразует число в шестнадцатеричный формат
15     # printf %s преобразует шестнадцатеричное значение в строку
16     letter=$(printf "%c" $(printf "%x" $((rand + 65)))>/dev/null)
17
18     result=$result$letter # Добавим букву к результату
19
20 done
21
22 echo "Случайная последовательность: $result" # Выводим результат

```

The command prompt shows the output of the script: "Случайная последовательность: 21". The prompt then returns to "PS C:\random.sh".

Рисунок 4.6: 6

## 5 Контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке: `1 while [$1 != «exit»]` В данной строчке допущены следующие ошибки: не хватает пробелов после первой скобки `[` и перед второй скобкой `]` выражение `$1` необходимо взять в `"`, потому что эта переменная может содержать пробелы Таким образом, правильный вариант должен выглядеть так:  
`while [ «$1» != «exit» ]`
2. Как объединить (конкатенация) несколько строк в одну? Чтобы объединить несколько строк в одну, можно воспользоваться несколькими способами: Первый: `VAR1=«Hello,» VAR2=" World" VAR3=«VAR1VAR2» echo «VAR3»` : *Hello, World : VAR1 = "Hello,"VAR1+ = "World"echo"VAR1»*  
Результат: *Hello, World*
3. Найдите информацию об утилите `seq`. Какими иными способами можно реализовать её функционал при программировании на `bash`? Команда `seq` в Linux используется для генерации чисел от ПЕРВОГО до ПОСЛЕДНЕГО шага INCREMENT. Параметры: `seq LAST`: если задан только один аргумент, он создает числа от 1 до LAST с шагом шага, равным 1. Если LAST меньше 1, значение `is` не выдает. `seq FIRST LAST`: когда заданы два аргумента, он генерирует числа от FIRST до LAST с шагом 1, равным 1. Если LAST меньше FIRST, он не выдает никаких выходных данных. `seq FIRST INCREMENT LAST`: когда заданы три аргумента, он генерирует числа от FIRST до LAST на шаге INCREMENT. Если LAST меньше, чем FIRST, он не

производит вывод. `seq -f «FORMAT» FIRST INCREMENT LAST`: эта команда используется для генерации последовательности в форматированном виде. `FIRST` и `INCREMENT` являются необязательными. `seq -s «STRING» ПЕРВЫЙ ВКЛЮЧЕНО`: Эта команда используется для `STRING` для разделения чисел. По умолчанию это значение равно `/n`. `FIRST` и `INCREMENT` являются необязательными. `seq -w FIRST INCREMENT LAST`: эта команда используется для выравнивания ширины путем заполнения начальными нулями. `FIRST` и `INCREMENT` являются необязательными.

4. Какой результат даст вычисление выражения  $\$(10/3)$ ? Результатом данного выражения  $\$(10/3)$  будет 3, потому что это целочисленное деление без остатка.
5. Укажите кратко основные отличия командной оболочки `zsh` от `bash`.  
Отличия командной оболочки `zsh` от `bash`: В `zsh` более быстрое автодополнение для `cd` с помощью `Tab` В `zsh` существует калькулятор `zcalc`, способный выполнять вычисления внутри терминала В `zsh` поддерживаются числа с плавающей запятой В `zsh` поддерживаются структуры данных «хэш» В `zsh` поддерживается раскрытие полного пути на основе неполных данных В `zsh` поддерживается замена части пути В `zsh` есть возможность отображать разделенный экран, такой же как разделенный экран `vim`
6. Проверьте, верен ли синтаксис данной конструкции `1 for ((a=1; a <= LIMIT; a++))` синтаксис данной конструкции верен, потому что, используя двойные круглые скобки, можно не писать `$` перед переменными `()`.
7. Сравните язык `bash` с какими-либо языками программирования. Какие преимущества у `bash` по сравнению с ними? Какие недостатки?  
Преимущества и недостатки скриптового языка `bash`: Один из самых

распространенных и ставится по умолчанию в большинстве дистрибутивах Linux, MacOS Удобное перенаправление ввода/вывода Большое количество команд для работы с файловыми системами Linux Можно писать собственные скрипты, упрощающие работу в Linux Недостатки скриптового языка bash: Дополнительные библиотеки других языков позволяют выполнить больше действий Bash не является языком общего назначения Утилиты, при выполнении скрипта, запускают свои процессы, которые, в свою очередь, отражаются на скорости выполнения этого скрипта Скрипты, написанные на bash, нельзя запустить на других операционных системах без дополнительных действий

## 6 Выводы

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

## **Список литературы**