

Шаблон отчёта по лабораторной работе номер 13

Дисциплина: Операционные системы

Крестененко Полина Александровна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	15

Список таблиц

Список иллюстраций

3.1	первый скрипт	7
3.2	проверка	8
3.3	первый скрипт измененный	8
3.4	первый скрипт измененный	9
3.5	проверка	9
3.6	проверка	9
3.7	справка	10
3.8	второй скрипт	10
3.9	необходимые команды	11
3.10	необходимые команды	11
3.11	необходимые команды	12
3.12	третий скрипт	12
3.13	проверка	13

1 Цель работы

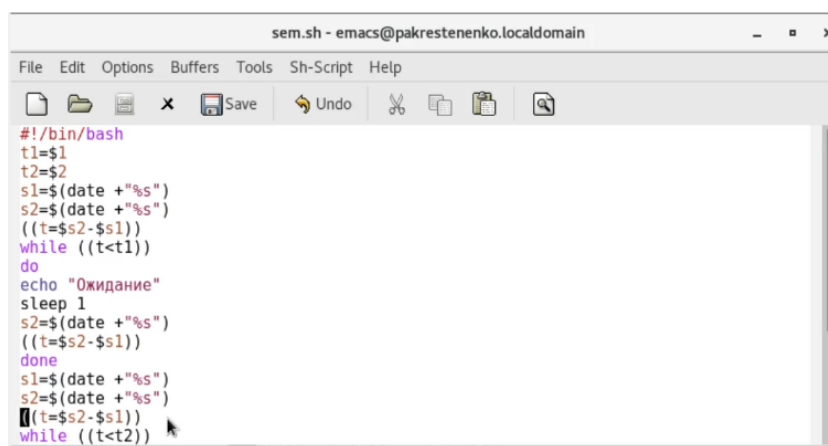
Изучить основы программирования в оболочке ОС UNIX. Научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

2 Задание

Выполнить три задания, представленные в тескте файла лабораторной работы.

3 Выполнение лабораторной работы

- 1) Написала 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). Для данной задачи я создала файл: sem.sh и написала соответствующий скрипт.(рис. -fig. 3.1).

The image shows a terminal window titled 'sem.sh - emacs@pakrestenenko.localdomain'. The window contains a bash script with the following content:

```
#!/bin/bash
t1=1
t2=2
s1=$(date +%s)
s2=$(date +%s)
((t=$s2-$s1))
while ((t<t1))
do
echo "Ожидание"
sleep 1
s2=$(date +%s)
((t=$s2-$s1))
done
s1=$(date +%s)
s2=$(date +%s)
((t=$s2-$s1))
while ((t<t2))
```

Рис. 3.1: первый скрипт

Далее я проверила работу написанного скрипта (команда «./sem.sh 4 7»), предварительно добавив право на исполнение файла (команда «chmod +x sem.sh») Скрипт работает корректно.(рис. -fig. 3.2).

```

[pakrestenenko@pakrestenenko ~]$ touch sem.sh
[pakrestenenko@pakrestenenko ~]$ emacs &
[1] 2304
[pakrestenenko@pakrestenenko ~]$ chmod +x sem.sh
[pakrestenenko@pakrestenenko ~]$ ./sem.sh 4 7
Ожидание
Ожидание
Ожидание
Ожидание
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение

```

Рис. 3.2: проверка

После этого я изменила скрипт так, чтобы его можно было выполнять в нескольких терминалах и проверила его работу (например, команда «./sem.sh 2 3 Ожидание > /dev/pts/1 &»)(рис. -fig. 3.3).

```

sem.sh - emacs@pakrestenenko.localdomain
File Edit Options Buffers Tools Sh-Script Help
[Icons: New, Open, Save, Undo, Cut, Copy, Paste, Find]
#!/bin/bash
function ogidanie
{
s1=$(date +%s)
s2=$(date +%s)
((t=s2-s1))
while ((t<t1))
do
echo "Ожидание"
sleep 1
s2=$(date +%s)
((t=s2-s1))
done
}
function vipolnenie
{
s1=$(date +%s)

```

Рис. 3.3: первый скрипт измененный

(рис. -fig. 3.4).


```

sem.sh - emacs@pakresteneko.localdomain
File Edit Options Buffers Tools Sh-Script Help
[Icons: File, Folder, Save, Undo, Cut, Copy, Paste, Find]

command=$3
while true
do
if [ "$command" == "Выход" ]
then
echo "Выход"
exit 0
fi
if [ "$command" == "Ожидание" ]
then ogidanie
fi
if [ "$command" == "Выполнение" ]
then vipolnenie
fi
echo "Следующее действие: "
read command
done

```

Рис. 3.4: первый скрипт измененный

(рис. -fig. 3.5).

```

[pakresteneko@pakresteneko ~]$ ./sem.sh 3 4 Ожидание > /dev/pts/2 &
[3] 2968
[pakresteneko@pakresteneko ~]$ ./sem.sh 3 5 Выполнение > /dev/pts/3 &
[4] 3058

[3]+  Stopped                  ./sem.sh 3 4 Ожидание > /dev/pts/2

```

Рис. 3.5: проверка

(рис. -fig. 3.6).

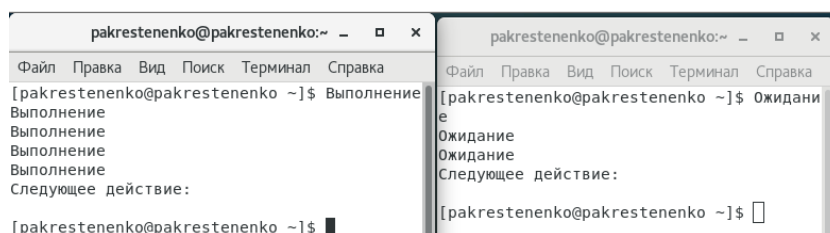


Рис. 3.6: проверка

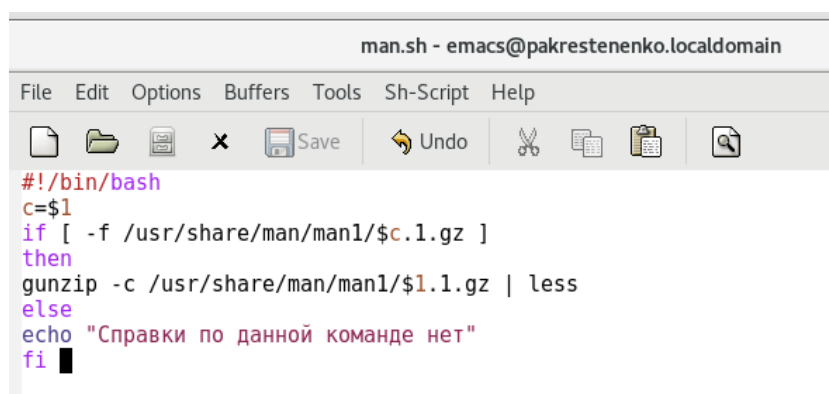
- 2) Реализовала команду man с помощью командного файла. Изучила содержимое каталога /usr/share/man/man1. В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд. Каждый архив можно открыть командой less сразу

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. -fig. 3.7).

```
[pakrestenenko@pakrestenenko man1]$ ls
.: 1.gz
[. 1.gz
a2p.1.gz
abrt-action-analyze-backtrace.1.gz
abrt-action-analyze-c.1.gz
abrt-action-analyze-ccpp-local.1.gz
abrt-action-analyze-core.1.gz
abrt-action-analyze-oops.1.gz
abrt-action-analyze-python.1.gz
abrt-action-analyze-vmcore.1.gz
abrt-action-analyze-vulnerability.1.gz
abrt-action-analyze-xorg.1.gz
abrt-action-check-oops-for-hw-error.1.gz
abrt-action-generate-backtrace.1.gz
abrt-action-generate-core-backtrace.1.gz
abrt-action-install-debuginfo.1.gz
abrt-action-list-dsos.1.gz
abrt-action-notify.1.gz
abrt-action-perform-ccpp-analysis.1.gz
abrt-action-save-kernel-data.1.gz
abrt-action-save-package-data.1.gz
abrt-action-trim-files.1.gz
abrt-applet.1.gz
```

Рис. 3.7: справка

Для данной задачи я создала файл: man.sh и написала соответствующий скрипт.
(рис. -fig. 3.8).



```
#!/bin/bash
c=$1
if [ -f /usr/share/man/man1/$c.1.gz ]
then
gunzip -c /usr/share/man/man1/$1.1.gz | less
else
echo "Справки по данной команде нет"
fi
```

Рис. 3.8: второй скрипт

Далее я проверила работу написанного скрипта (команды «./man.sh ls» и «./man.sh mkdir»), предварительно добавив право на исполнение файла (команда «chmod +x man.sh») Скрипт работает корректно.(рис. -fig. 3.9).

```
[pakrestenenko@pakrestenenko ~]$ chmod +x man.sh
[pakrestenenko@pakrestenenko ~]$ ./man.sh ls
[pakrestenenko@pakrestenenko ~]$ ./man.sh mkdir
[1]+  Done                  emacs
[pakrestenenko@pakrestenenko ~]$ █
```

Рис. 3.9: необходимые команды

(рис. -fig. 3.10).

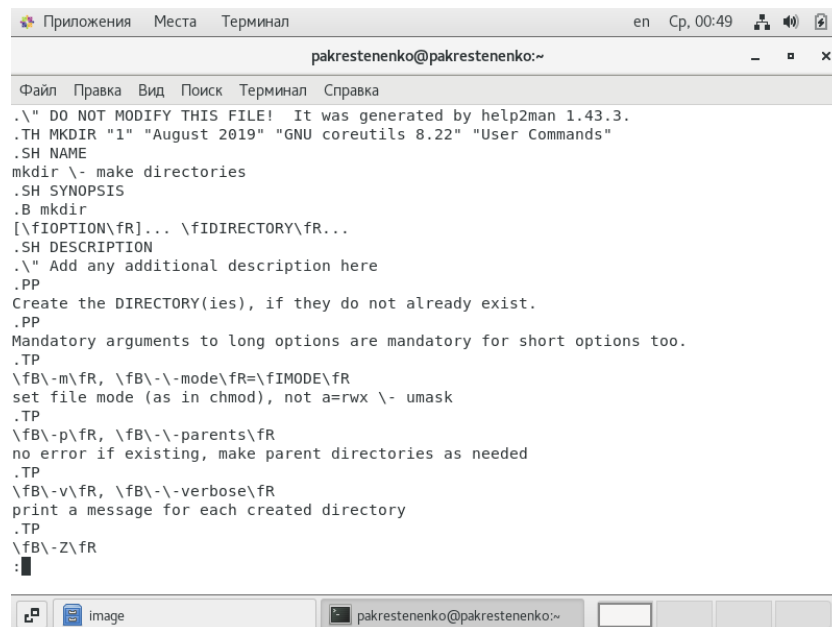


Рис. 3.10: необходимые команды

(рис. -fig. 3.11).

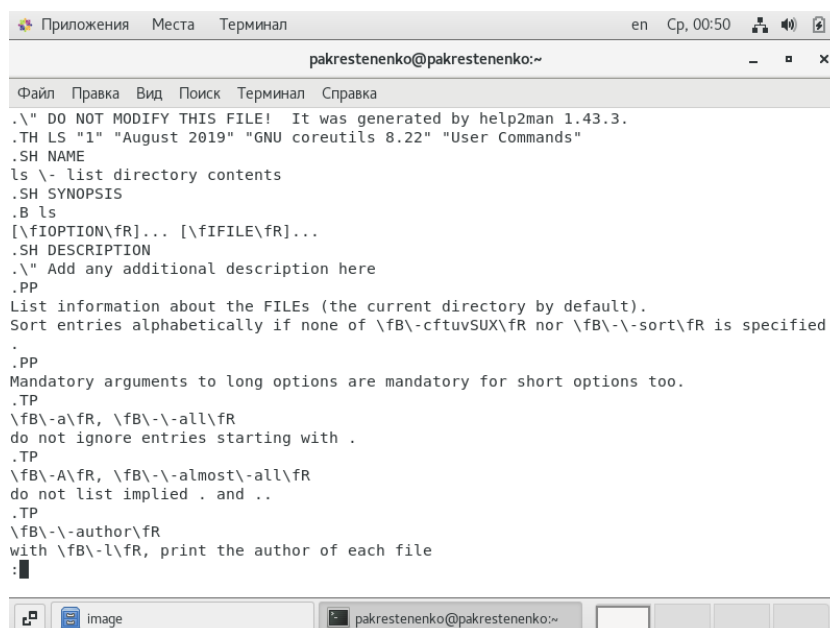


Рис. 3.11: необходимые команды

- 3) Используя встроенную переменную \$RANDOM, написала командный файл, генерирующий случайную последовательность букв латинского алфавита. Для данной задачи я создала файл: random.sh и написала соответствующий скрипт.(рис. -fig. 3.12).

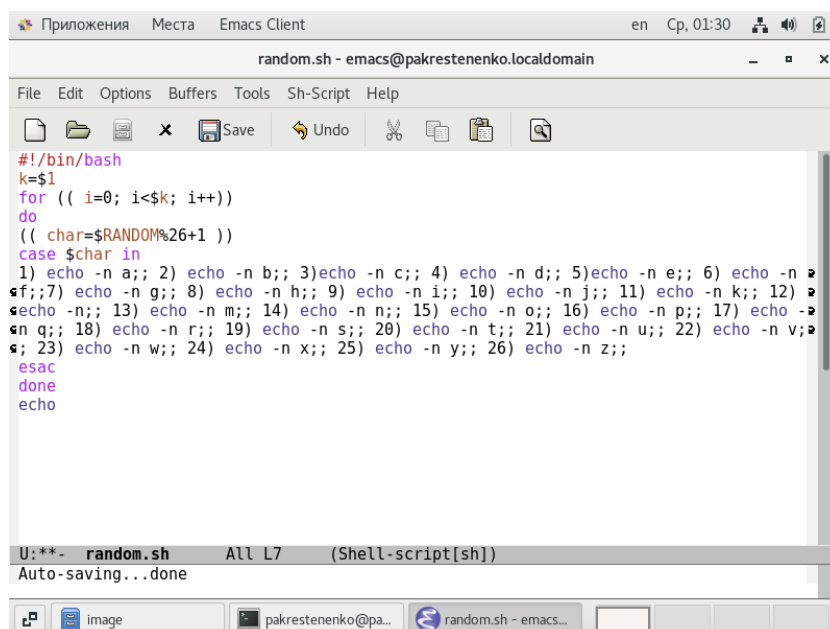


Рис. 3.12: третий скрипт

Далее я проверила работу написанного скрипта (команды «./random.sh 7» и «./random.sh 15»), предварительно добавив право на исполнение файла (команда «chmod +x random.sh») Скрипт работает корректно. (рис. -fig. 3.13).

```
[pakrestenenko@pakrestenenko ~]$ touch random.sh
[pakrestenenko@pakrestenenko ~]$ emacs &
[1] 4179
[pakrestenenko@pakrestenenko ~]$ chmod +x random.sh
[pakrestenenko@pakrestenenko ~]$ ./random.sh 7
nfhnrtr
[pakrestenenko@pakrestenenko ~]$ ./random.sh 16
vxqkpxodxecmohpx
[pakrestenenko@pakrestenenko ~]$ ./random.sh 26
ipfaftgsuhxtjxetepwnhxpej
-
```

Рис. 3.13: проверка

Контрольные вопросы: 1) В данной строчке допущены следующие ошибки: не хватает пробелов после первой скобки [и перед второй скобкой] выражение 1 “”, 2), : : VAR1 = "Hello,"VAR2 = "World"VAR3 = "VAR1VAR2"echo"VAR3" Результат: Hello, World Второй: VAR1="Hello," VAR1+="World" echo "\$VAR1" Результат: Hello, World 3) Команда seq в Linux используется для генерации от ПЕРВОГО до ПОСЛЕДНЕГО шага INCREMENT. чиселПараметры: seq LAST: если задан только один аргумент, он создает числа от 1 о 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 форматированном виде. последовательности в 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) Результатом данного выражения $\$((10/3))$ будет 3, потому что это целочисленное деление без остатка. 5) 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) for ((a=1; a <= LIMIT; a++)) синтаксис данной конструкции верен, потому что, используя двойные круглые скобки, можно не писать \$ перед переменными (). 7) Преимущества 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

4 Выводы

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