# РОCСИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

## Факультет физико-математических и естественных наук

## Кафедра прикладной информатики и теории вероятностей

## ОТЧЕТ

## ПО ЛАБОРАТОРНОЙ РАБОТЕ № 13

### *дисциплина: Операционные системы*

Студент: Губина Ольга Вячеславовна

Группа: НПИбд-01-20

Преподаватель: Велиева Татьяна Рефатовна

МОСКВА

2021 г.

### Цель работы:

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

### Задачи:

1. Научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов;
2. Применить их на практике.

### Теоретическое введение:

В данной лабораторной работе нам предстоит научиться писать командные файлы и использовать их на практике. Для этого нам необходимо ознакимиться с некоторой теорией.

**Командные процессоры (оболочки)**

*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;
* С-оболочка (или csh) — надстройка на оболочкой Борна, использующая Сподобный синтаксис команд с возможностью сохранения истории выполнения команд;
* оболочка Корна (или ksh) — напоминает оболочку С, но операторы управления программой совместимы с операторами оболочки Борна;
* BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек С 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/Linuxподобных 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.

**Переменные в языке программирования bash**

Командный процессор *bash* обеспечивает возможность использования переменных типа строка символов. Имена переменных могут быть выбраны пользователем. Пользователь имеет возможность присвоить переменной значение некоторой строки символов. Например, команда

mark=/usr/andy/bin

присваивает значение строки символов /usr/andy/bin переменной mark типа строка символов.

Использование:

mv afile ${mark}

переместит файл afile из текущего каталога в каталог с абсолютным полным именем /usr/andy/bin.

Использование значения, присвоенного некоторой переменной, называется *подстановкой*.

**Команды read и echo**

Команда read позволяет записать значение для переменной с клавиатуры. Она имеет следующий синтаксис:

read <variable>

Команда echo выводит текст на экран, если имеет вид:

echo "Some text"

В данном случае она выведет на экран *Some text*.

С помощью данной команды также можно вывести на экран содержимое, например, переменных:

echo <variable>

С прочей теорией и основами языка bash можно ознакомиться в материалах к *лабораторной работе №11*[1].

Также в ходе выполнения заданий лабораторной работы я столкнулась в необходимости изучения дополнительных натериалов, а именно:

* циклы if[2]
* массивы[3]
* утилита test[4]

### Выполнение работы:

***Задание 1***

*Написать командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени t2<>t1, 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.*

Создадим командный файл lock.sh, который будет релаизовывать упрощенный механизм семафоров, с помощтю команды vi lock.sh, он сразу же откроется, начнем его написание (*рисунок 1*).

Задаем переменную lock, в которой хранится полный путь к файлу, который мы будем блокировать (даже если такого файла не существует, он будет создан по средствам последующих команд, выполняющихся в ком андном файле).

Далее присваиваем через команду exec {fn}>$lock дескриптор, для возможности работы с командой flock в дальнейшем.

Входим в бесконечный цикл while, условием которого является то, что lock является обычным файлом - test -f. В данном цикле имеется условие if, если файл уже заблокирован - flock -n ${fn} (где ${fn} дескриптор, номер нашего файла), - выводим сообщение об этом и выжидаем 4 секунды, имитируя внутреннюю рботу с файлом. После этого разблокируем файл и выводим сообщение об этом. Если же файл заблокировать не получается, тоже выводим об этом сообщение.

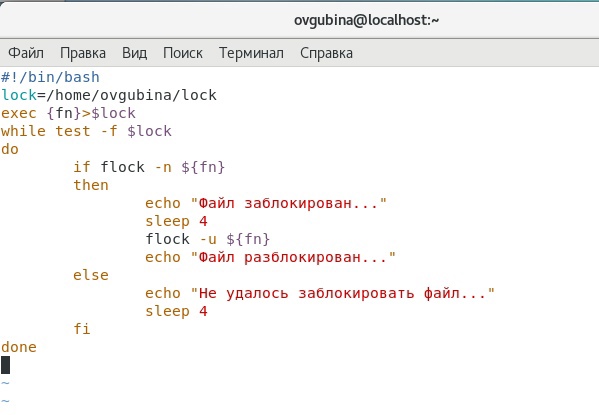


рисунок 1: командный файл lock.sh

Теперь проверим правильность работы нашего командного файла (*риснуок 2*). Для этого сначала добавим права на выполнение chmod +x lock.sh.



рисунок 2: присваиваем возможность запуска

Теперь откроем текстовую консоль tty2 нажатием клавиш Ctrl + Alt + F2. В ней вызовем наш командный файл и переадресуем вывод в третью текстовую консоль ./lock.sh > /dev/tty3 (*риснуок 3*).



рисунок 3: переадресация вывода

Открываем консоль tty3 и видим, что в ней действительно выполняется командный файл - файл блокируется и разблокируется (*рисунок 4*).

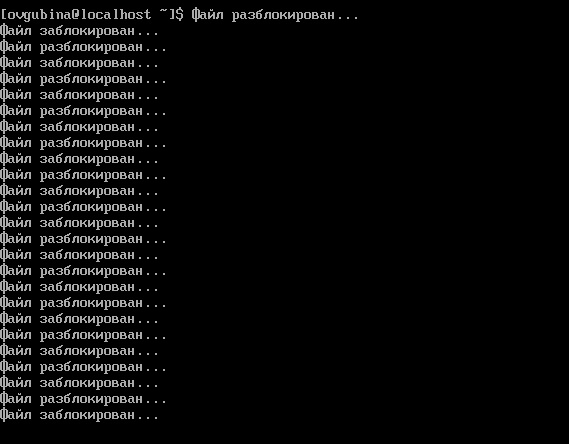


рисунок 4: работа в 3 текстовой консоли

Запустим наш файл в консоли tty4 в фоновом режиме ./lock.sh &(*рисунок 5*).

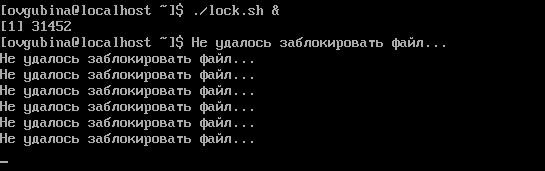


рисунок 5: 4 текстовая консоль

Таким образом мы наблюдаем, как запущенный файл в привилегированном режиме блокирует файл, работает с ним и разблокирует, а запущенный в фоновом режиме элементарно не успевает что-либо сделать с файлом, поскольку время его заплонированнной блокировки совпадает со временем, когда привилегировнный запуск работает с файлом.

***Задание 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.*

Создадим командный файл man13.sh, который будет релаизовывать команду man, с помощтю команды vi man13.sh, он сразу же откроется, начнем его написание (*рисунок 6*).

2 строка - переходим в каталог /usr/share/man/man1. В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд, которые мы будем просматривать. Зададим переменную command - имя команды, которое мы будем вводить с клавиатуры. Далее на экран командой echo выведем сообщение о том, что нам необходимо ввести имя команды, информацию о которой мы хотим узнать. Вводим ее с клавиатуры через read и снова выводим сообщение о том, что далее будет показана информация по данной команде (однако это ненужно, поскольку информация будет показана не в терминале, а в отдельном окне как привызове man). Командой less просмотрим содержимое справки, используя указатель на имя команды $.

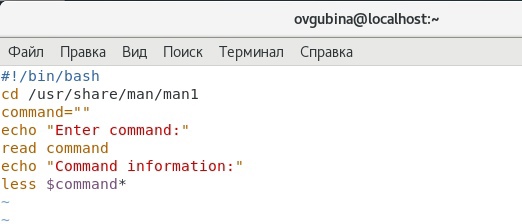


рисунок 6: командный файл man13.sh

Теперь проверим правильность работы нашего командного файла (*риснуок 7*). Для этого сначала добавим права на выполнение chmod +x man13.sh. Далее вызовем наш файл для проверки в качестве команды ./man13.sh. Видим, что он выводит все сообщения так, как было задумано, после ввода команды (мы будем просматривать информацию о команде cp), действительно выводит справку о ней (*рисунок 8*). Нажимаем клавишу q, чтобы закончить просмотр справки.

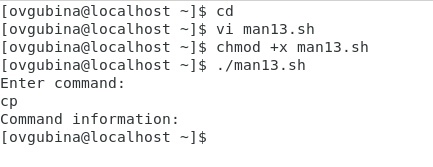


рисунок 7: работа командного файла man13.sh

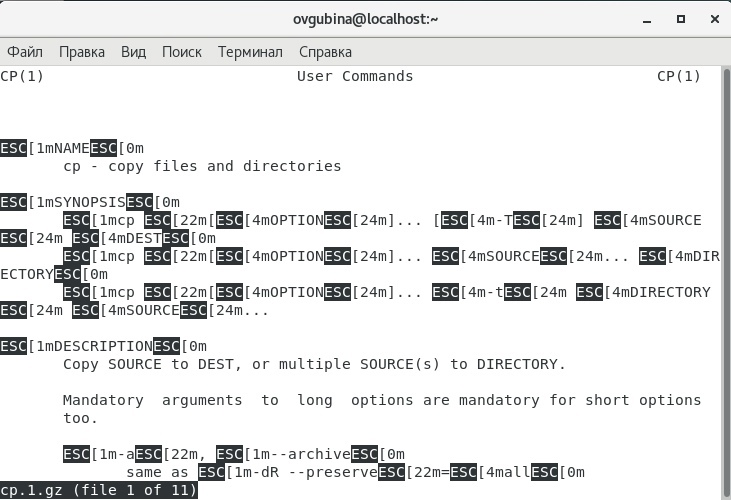


рисунок 8: вывод справки man

***Задание 3***

*Используя встроенную переменную RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767.*

Создадим командный файл random\_line.sh, который будет релаизовывать генерацию строк с рандомным надором букв латинского алфавита, с помощью команды vi random\_line.sh, он сразу же откроется, начнем его написание (*рисунок 9*).

Сначала зададим переменную n, котрая будет обозначать длину генерируемой строки, она также задается рандомно, однако выбираем диапозон случайных значений от 1 до 50, чтобы строка не была сильно длинной. Обозначим переменнную r, в нее будет записывааться рандомный номер элемента массива, о котором мы сейчас поговорим.

Объявим массив, в который запишем все 26 букв латинского алфавита в качестве его элементов (порядковые номера начинаются с 0).

После объявления массива входим в цикл for, который выполняется n-1 раз. Для каждого прохода цикла перменной r присваиваем рандомное значение в диапозоне от 0 до 25, т.е. номера элементов нашего массива. Далее выводим на экран элемент массива с таким номером - букву алфавита - используем при этом ключ -n, который съедает перенос строки при выводе на экран, таким образом буквы будут записываться в одну строчку. Однако, чтобы не прилепить следующую новую строку терминала к нашей строке, мы и взяли цикл с n-1 проходов. Поэтому n-ый раз генерируем букву уже все цикла, она является последней, и здесь уже используем вывод с переносом строки - без опций.

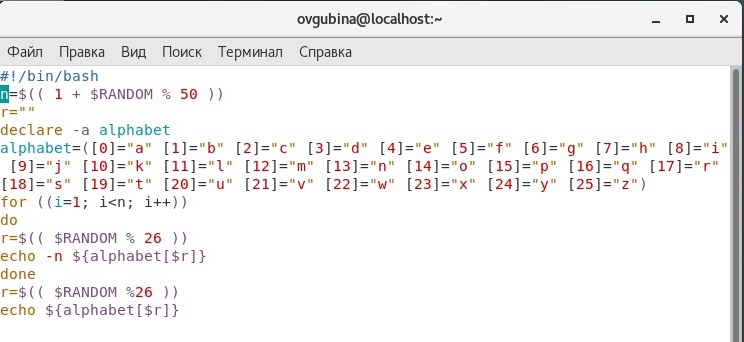


рисунок 9: командный файл random\_line.sh

Проверим работу нашего командного файла (*рисунок 5*). Для этого сначала добавим права на выполнение chmod +x random\_line.sh. Далее вызовем наш файл для проверки в качестве команды ./random\_line.sh. Повторим это несколько раз, видим, что на выход получаем строки разной длинны с произвольным набором букв - задача выполнена.

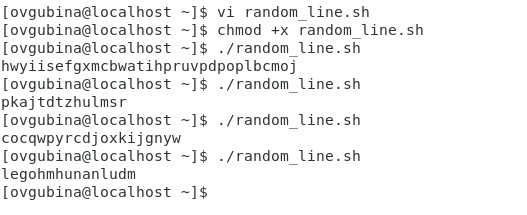


рисунок 10: работа командного файла random\_line.sh

## Вывод:

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

### Библиография:

[1] [Лабораторная работа №11](https://esystem.rudn.ru/pluginfile.php/1142377/mod\_resource/content/2/008-lab\_shell\_prog\_1.pdf)

[2] [Циклы if](https://habr.com/ru/company/ruvds/blog/325928/)

[3] [Использование массивов в bash](https://losst.ru/massivy-bash)

[4] [Утилита test](https://ru.wikipedia.org/wiki/Test)