

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 13

дисциплина: Операционные системы

Студент: Ким Реачна

Группа: НПИбд-02-20

Москва

2021г.

Цель работы:

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

Объект и предмет исследования:

Программирование в оболочке **ОС UNIX/Linux**

Теоретическое введение:

Циклы Bash:

Циклы позволяют выполнять один и тот же участок кода необходимое количество раз. В большинстве языков программирования существует несколько типов циклов. Большинство из них поддерживаются оболочкой Bash.

- `for` - позволяет перебрать все элементы из массива или использует переменную `count` для определения количества повторений
- `while` - цикл выполняется пока условие истинно
- `until` - цикл выполняется пока условие ложно.

Другие команды:

flock — утилита, которая позволяет использовать лок-файл для предотвращения запуска копии процесса (вашего скрипта, крона или чего-то еще).

sleep — одна из самых простых команд. Как видно из названия, его единственная функция — спать. Другими словами, он вводит задержку на указанное время.

\$RANDOM - возвращает псевдослучайные целые числа в диапазоне 0 - 32767.

Условные обозначения и символы:

- `$*` — отображается вся командная строка или параметры оболочки;
- `$?` — код завершения последней выполненной команды;
- `$$` — уникальный идентификатор процесса, в рамках которого выполняется командный процессор;
- `#!` — номер процесса, в рамках которого выполняется последняя вызванная на выполнение в командном режиме команда;
- `$-` — значение флагов командного процессора;
- `${#*}` — возвращает целое число — количество слов, которые были результатом `$*`;
- `${#name}` — возвращает целое значение длины строки в переменной `name`;
- `${name[n]}` — обращение к `n`-му элементу массива;
- `${name[*]}` — перечисляет все элементы массива, разделённые пробелом;
- `${name[@]}` — то же самое, но позволяет учитывать символы пробелы в самих переменных;
- `${name:-value}` — если значение переменной `name` не определено, то оно будет заменено на указанное `value`;
- `${name:value}` — проверяется факт существования переменной;
- `${name=value}` — если `name` не определено, то ему присваивается значение `value`;
- `${name?value}` — останавливает выполнение, если имя переменной не определено, и выводит `value` как сообщение об ошибке;
- `name + value` — это выражение работает противоположно `{name-value}` Если переменная определена, то подставляется `value`;
- `${name#pattern}` — представляет значение переменной `name` с удалённым самым коротким левым образцом (`pattern`);

- `${#name[*]}` и `${#name[@]}` — эти выражения возвращают количество элементов в массиве `name`.

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. Доработать программу так, чтобы имелась возможность взаимодействия трёх и более процессов.

1. Создайте новый командный файл с именем `sema.sh` использование редактора `vi` (`vi sema.sh`) и написать программу, которая выполняет необходимые в нашей работе действия (*Рисунок 1*).

Рисунок 1: Создать командный файл `sema.sh`

```
kim@kim-VirtualBox: ~$ vi sema.sh

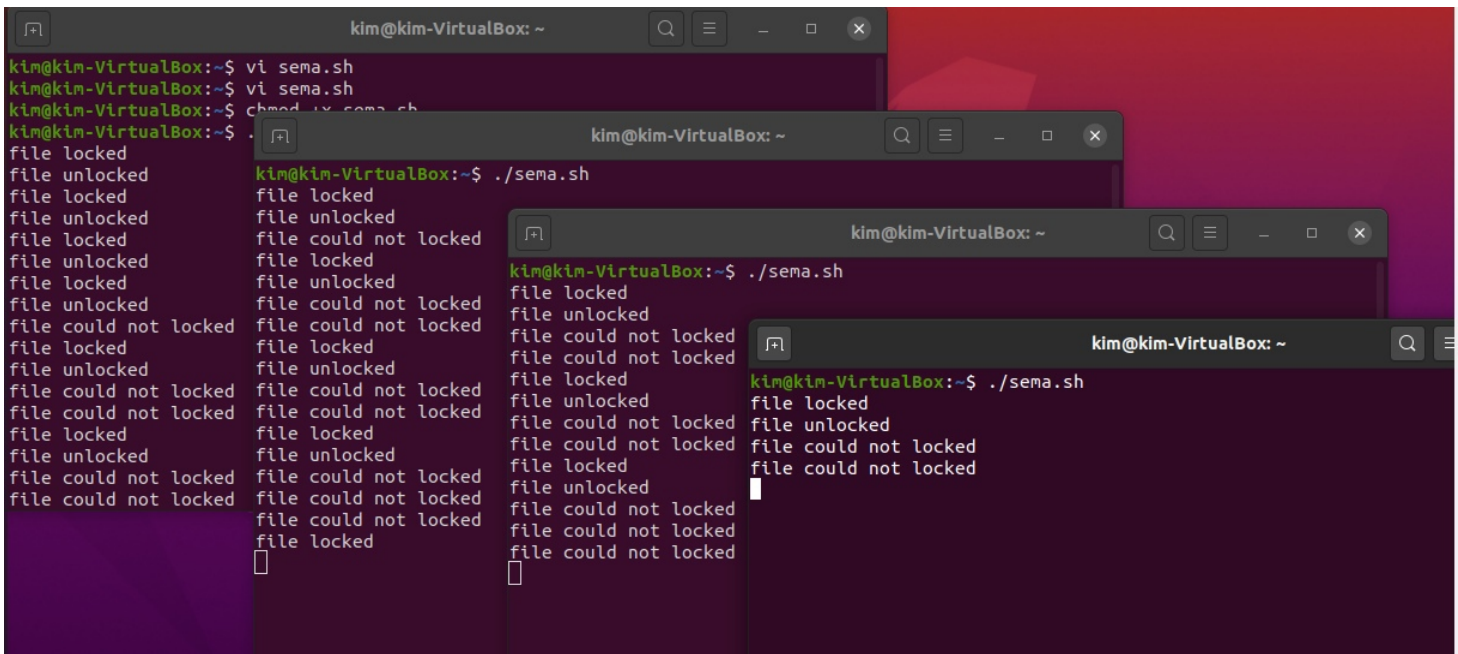
#!/bin/bash
exec {fn}>./lock.file
while test -f ./lock.file
do
if flock -n ${fn}
then
echo "file locked"
sleep 5
flock -u ${fn}
echo "file unlocked"
sleep 1
else
echo "file could not locked"
sleep 5
fi
done

~
~
~
"sema.sh" 17L, 209C 14,1 All
```

Структура программы: Сначала мы присваиваем файлу номер, а затем в цикле пытаемся получить доступ к файлу. Условие цикла, для которого выполняется `while test -f ./lock.file`. Это условие верно, если файл существует. Затем с условием `if flock -n ${fn}` в этом условии мы проверяем, заблокирован ли файл. Затем отобразите информацию об этом, в противном случае мы заблокируем файл и отобразим информацию о нем, а затем подождем 5 секунд в режиме `sleep 5` и разблокируем файл.

2. Теперь давайте запустим команду `chmod +x sema.sh` чтобы выполнить файл, затем откройте несколько терминалов и выполните нашу команду `./sema.sh` в каждом из них и пусть наблюдают за ними. Как мы видим результат, каждый из терминалов пытается получить доступ к файлам (Рисунок 2).

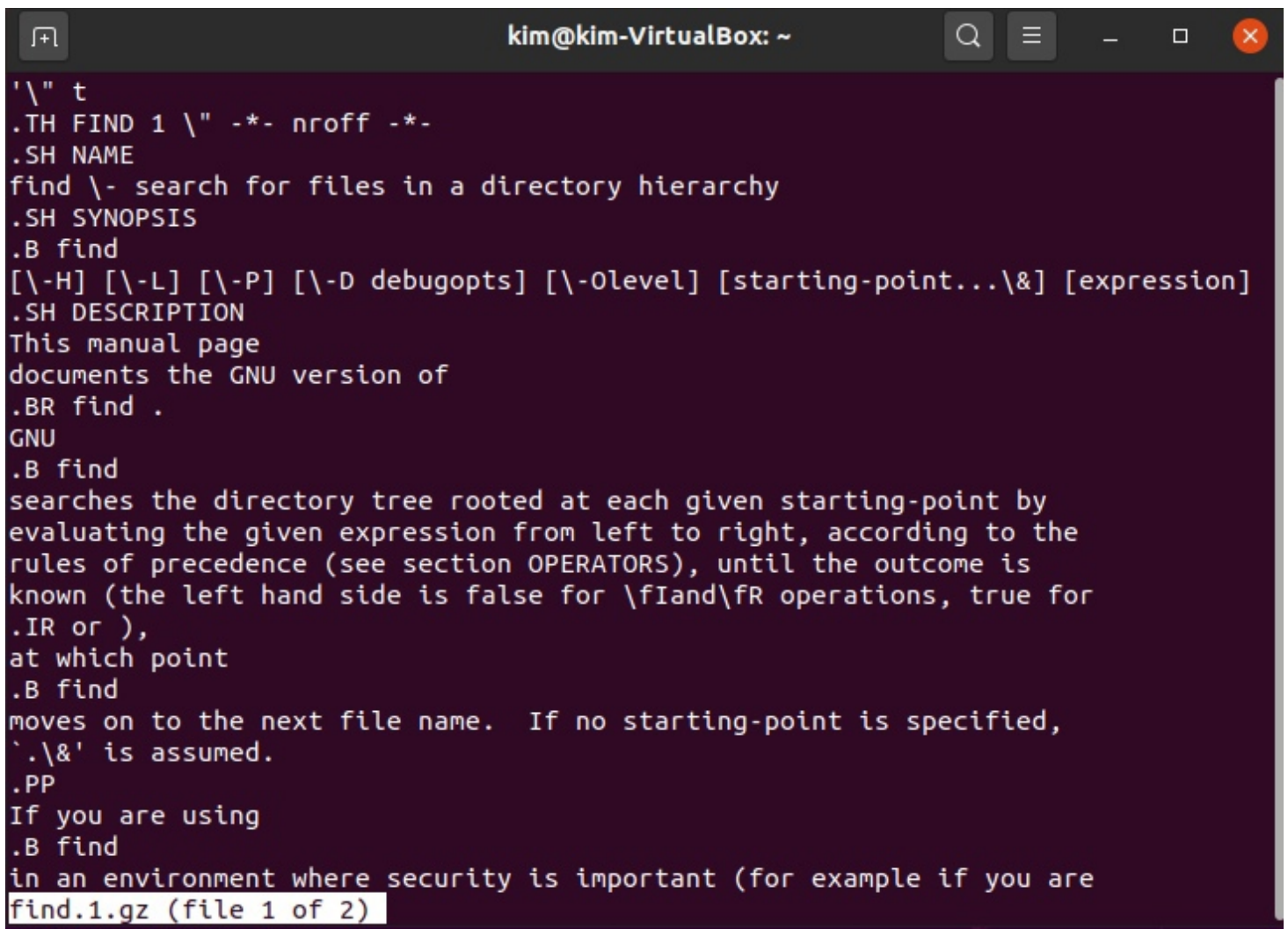
Рисунок 2:Проверка работы



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`.

1. Сначала давайте создадим другой новый командный файл `man.sh` снова используя редактор `vi man.sh` и написать в нем программу. Во второй строке мы пишем `/usr/share/man/man1`, чтобы найти в нем файл руководства и открыть окно предварительного просмотра с помощью команды `less` (Рисунок 3).

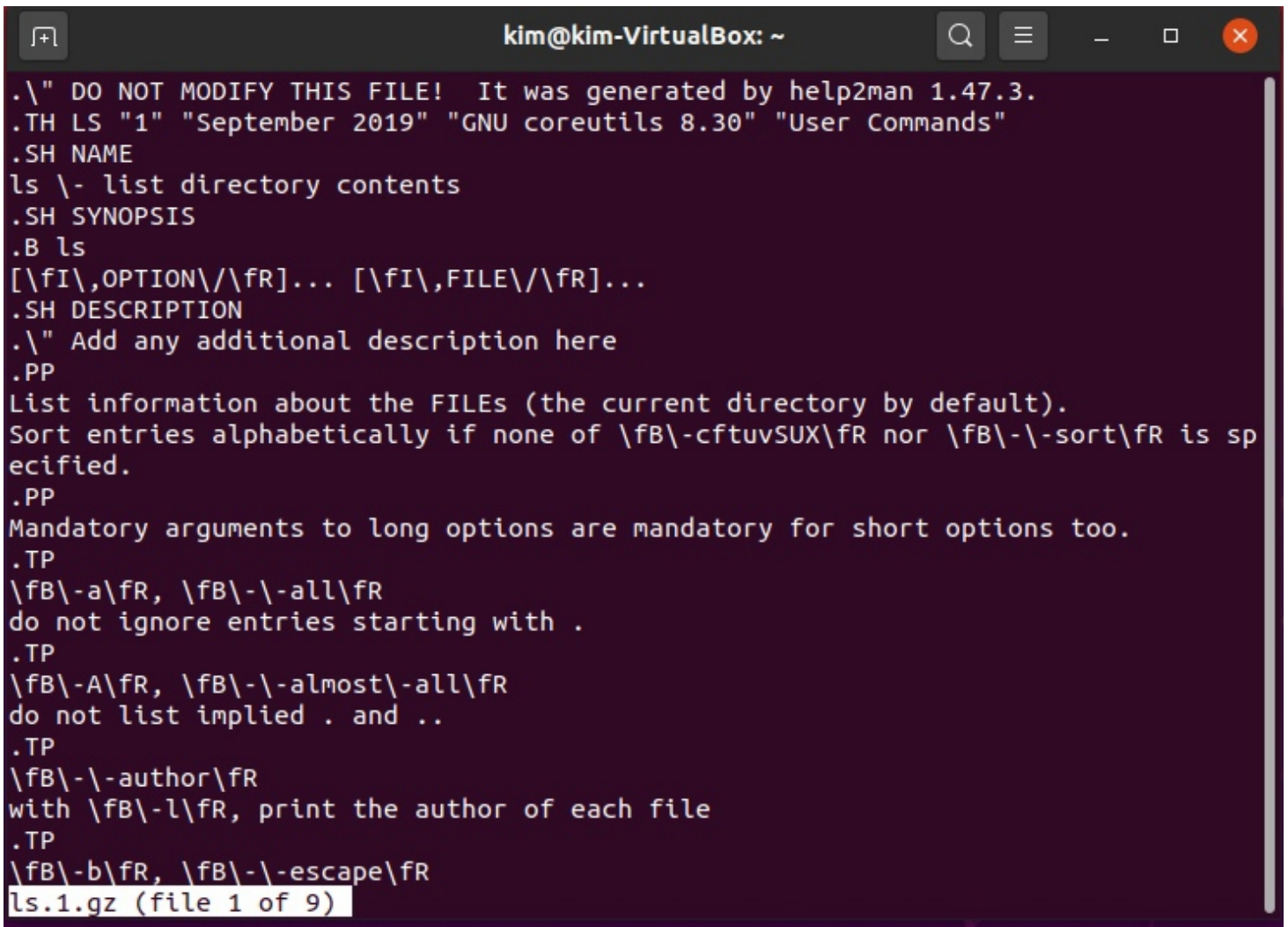
Рисунок 3: Создать новый файл `man.sh`



The image shows a terminal window titled "kim@kim-VirtualBox: ~". The terminal displays the manual page for the 'find' command. The text is as follows:

```
'\" t
.TH FIND 1 \" -*- nroff -*-
.SH NAME
find \- search for files in a directory hierarchy
.SH SYNOPSIS
.B find
[\\-H] [\\-L] [\\-P] [\\-D debugopts] [\\-Olevel] [starting-point...\\&] [expression]
.SH DESCRIPTION
This manual page
documents the GNU version of
.BR find .
GNU
.B find
searches the directory tree rooted at each given starting-point by
evaluating the given expression from left to right, according to the
rules of precedence (see section OPERATORS), until the outcome is
known (the left hand side is false for \\fIand\\fR operations, true for
.IR or ),
at which point
.B find
moves on to the next file name. If no starting-point is specified,
'\\.\\&' is assumed.
.PP
If you are using
.B find
in an environment where security is important (for example if you are
find.1.gz (file 1 of 2)
```

Рисунок 6: Проверка работы



```
kim@kim-VirtualBox: ~  
.\ " DO NOT MODIFY THIS FILE! It was generated by help2man 1.47.3.  
.TH LS "1" "September 2019" "GNU coreutils 8.30" "User Commands"  
.SH NAME  
ls \- list directory contents  
.SH SYNOPSIS  
.B ls  
[\fI\,OPTION\ /\fR]... [\fI\,FILE\ /\fR]...  
.SH DESCRIPTION  
.\ " Add any additional description here  
.PP  
List information about the FILES (the current directory by default).  
Sort entries alphabetically if none of \fB\ -cftuvSUX \fR nor \fB\ -\ -sort \fR is sp  
ecified.  
.PP  
Mandatory arguments to long options are mandatory for short options too.  
.TP  
\fB\ -a \fR, \fB\ -\ -all \fR  
do not ignore entries starting with .  
.TP  
\fB\ -A \fR, \fB\ -\ -almost -all \fR  
do not list implied . and ..  
.TP  
\fB\ -\ -author \fR  
with \fB\ -l \fR, print the author of each file  
.TP  
\fB\ -b \fR, \fB\ -\ -escape \fR  
ls.1.gz (file 1 of 9)
```

Задание 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.

1. Создайте командный файл с именем `random.sh` и написать в нем программу. Первую строку мы называем `/bin/bash`, затем представляем `n` переменных, затем мы для цикла от 1 до `n`, затем создаем случайное число `echo -n $RANDOM` и заменяем все числа на буквы `tr ' [0-9] ' '[a-z] ' .` (Рисунок 7).

Рисунок 7: Создать новый файл `random.sh`


```
kim@kim-VirtualBox:~$ vi random.sh

#!/bin/bash
n=$((1+$RANDOM%3))
for ((i=1; i<n; i++))
do
echo -n $RANDOM | tr '[0-9]' '[a-z]'
done
echo $RANDOM |tr '[0-9]' '[a-z]'

~
"random.sh" 13L, 137C 9,0-1 All
```

2. Теперь давайте проверим нашу работу с помощью той же команды `chmod +x random.sh` тогда `./random.sh` чтобы проверить это, как мы видим, в результатах отображается случайный набор букв разной длины.

Рисунок 8: Проверка работы

```
kim@kim-VirtualBox:~$ chmod +x random.sh
kim@kim-VirtualBox:~$ ./random.sh
cdfff
kim@kim-VirtualBox:~$ ./random.sh
ccafffcicbi
kim@kim-VirtualBox:~$ ./random.sh
ejcb
kim@kim-VirtualBox:~$ ./random.sh
bbbei
kim@kim-VirtualBox:~$ ./random.sh
bibbihcha
kim@kim-VirtualBox:~$ ./random.sh
cdiba
kim@kim-VirtualBox:~$
```

Контрольные вопросы:

Нужно взять в кавычки «\$1».

2. Написать переменные одну за другой. Например: `A = "BC"` . Либо с помощью оператора `+=`. Например: `B += C [3]`

3. Эта утилита выводит последовательность целых чисел с заданным шагом.

Также можно реализовать с помощью утилиты `jot`.

4. 3

5. В `zsh` можно настроить отдельные сочетания клавиш так, как вам нравится.

Использование истории команд в `zsh` ничем особенным не отличается от `bash`. `zsh` очень удобен для повседневной работы и делает добрую половину рутины за вас. Но стоит обратить внимание на различия между этими двумя оболочками. Например, в `zsh` после `for` обязательно вставлять пробел, нумерация массивов в `zsh` начинается 1, чего совершенно невозможно понять. Так, если вы используете shell для повседневной работы, исключаящей написание скриптов, используйте `zsh`. Если вам часто приходится писать свои скрипты, только `bash`! Впрочем, можно комбинировать. Как установить `zsh` в качестве оболочки по умолчанию для отдельного пользователя: `o`.

6. Синтаксис верен.

7. Преимущества:

- По сравнению с `cmd` у `bash` больше возможностей.
 - По сравнению с нескриптовыми языками программирования у него более низкий порог вхождения.
 - Его не нужно отдельно устанавливать, он встроен в операционную систему.
- Недостатки
- В интернете меньше дополнительной информации про него, чем про языки программирования.
 - Сложнее отлаживать программу.

Вывод:

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

Библиография:

[1]:[Лабораторная №13](#)

[2]:[команда man linux](#)

[3]:[страница руководства по поиску](#)