

Лабораторная работа номер 12

Программирование в командном процессоре ОС UNIX. Расширенное
программирование

Мальков Роман

Содержание

Цель работы	3
Задание	4
Ход работы	5
Выводы	9

Цель работы

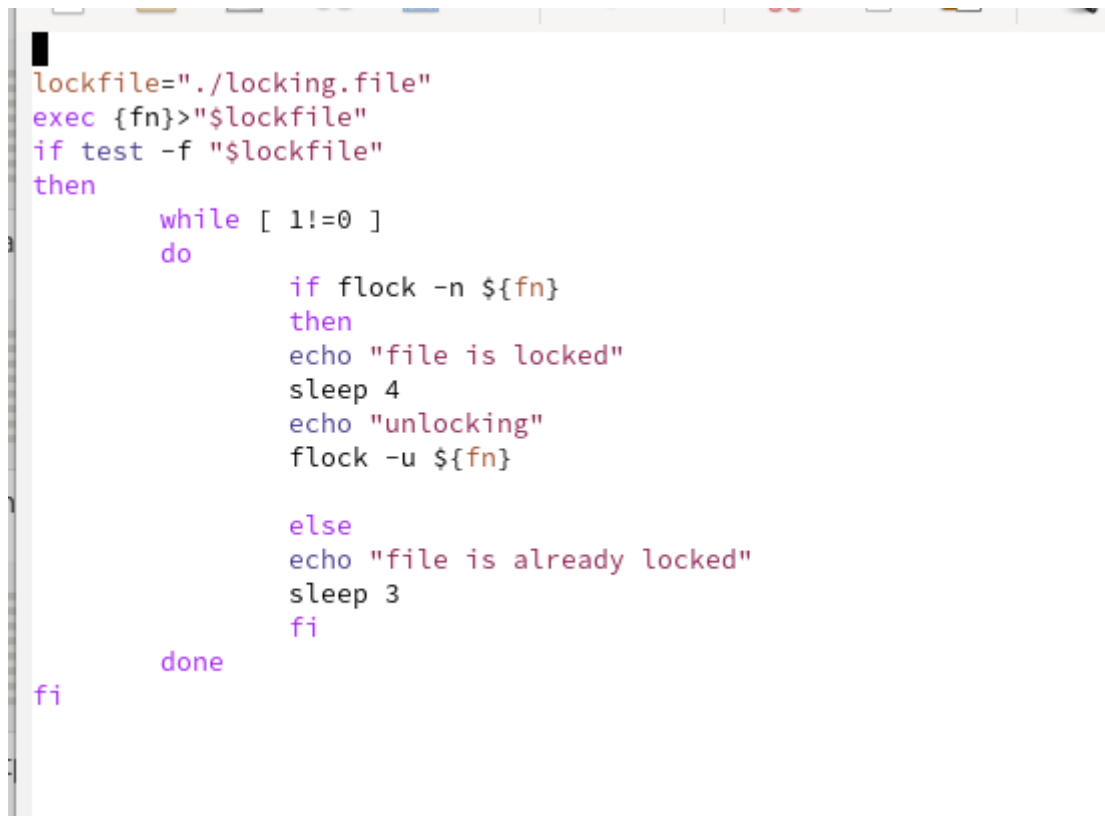
Изучить основы программирования в оболочке ОС UNIX. Научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

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 имела 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.

Ход работы

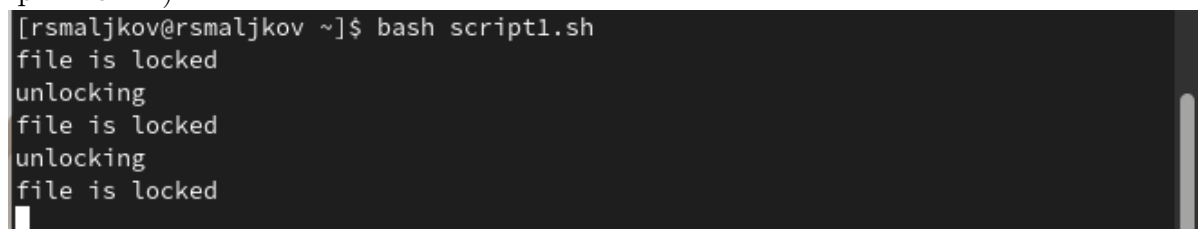
1. Код и результаты выполнения смотреть в скриншотах 1-3.



```
lockfile="./locking.file"
exec {fn}>"$lockfile"
if test -f "$lockfile"
then
    while [ 1!=0 ]
    do
        if flock -n ${fn}
        then
            echo "file is locked"
            sleep 4
            echo "unlocking"
            flock -u ${fn}

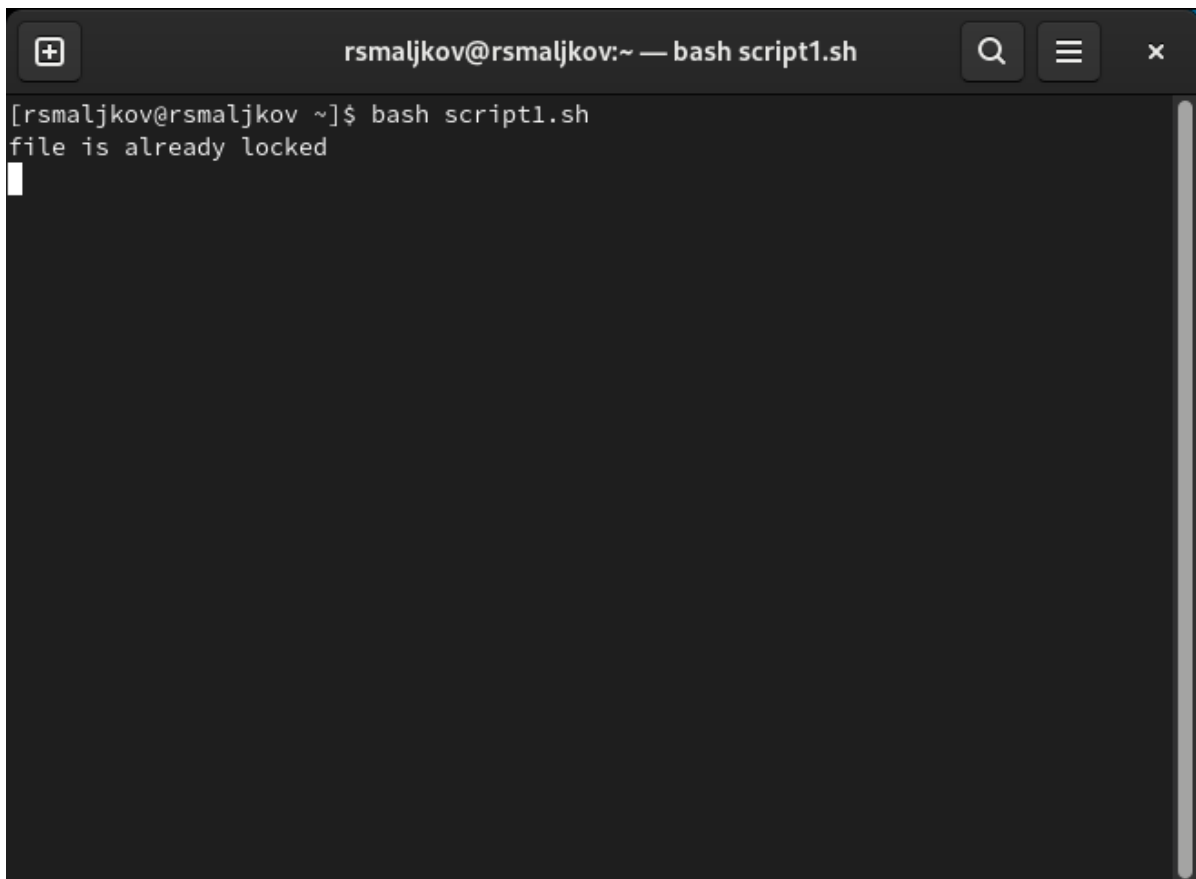
        else
            echo "file is already locked"
            sleep 3
        fi
    done
fi
```

Скриншот 1)



```
[rsmaljkov@rsmaljkov ~]$ bash script1.sh
file is locked
unlocking
file is locked
unlocking
file is locked
```

(Скриншот 2)



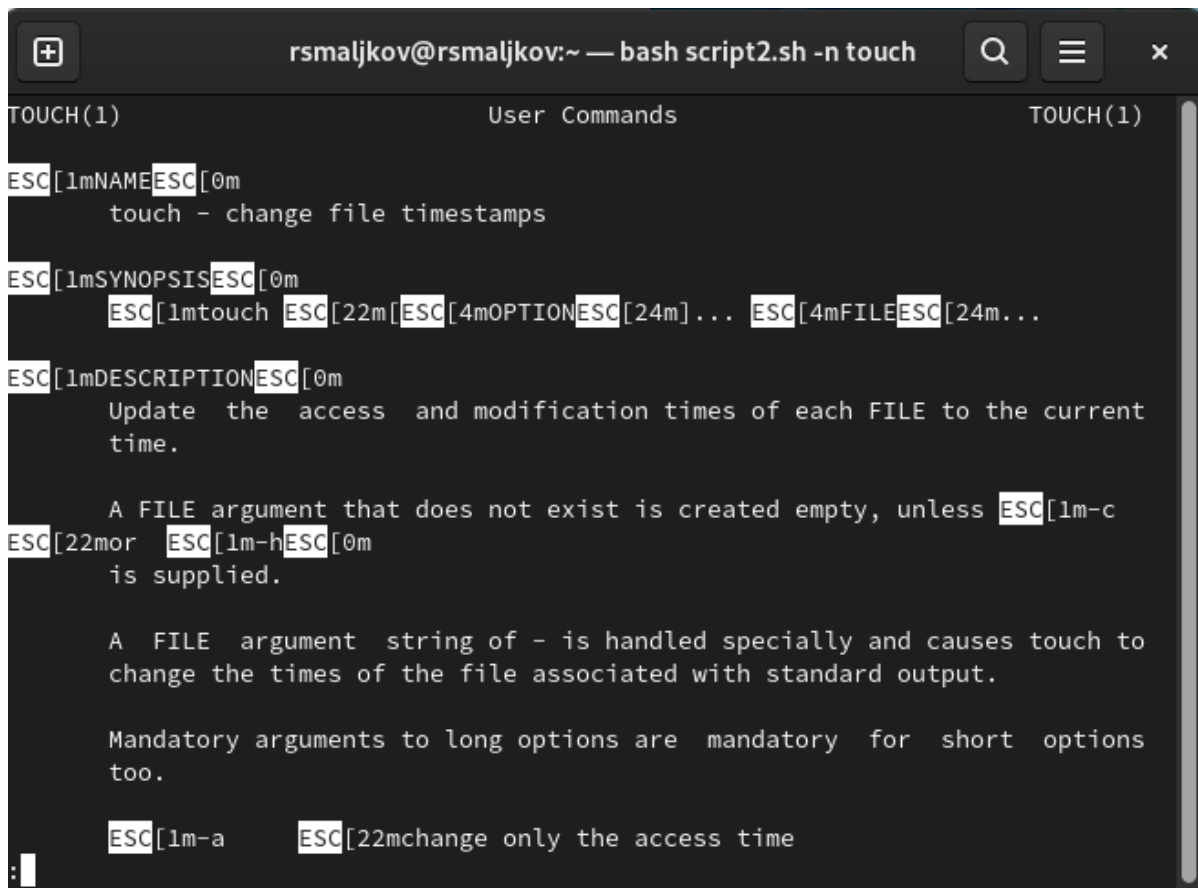
(Скриншот 3)

2. Код и результаты выполнения смотреть в скриншотах 4-5.

```
1 command = ""
2
3 while getopts :n: opt
4 do
5 case $opt in
6 n) command="$OPTARG";;
7 esac
8 done
9
10 if test -f "/usr/share/man/man1/$command.1.gz"
11 then less /usr/share/man/man1/$command.1.gz
12 else
13 echo "no such command"
14 fi
```

(Скриншот

4)



```
TOUCH(1) User Commands TOUCH(1)
ESC[1mNAMEESC[0m
    touch - change file timestamps
ESC[1mSYNOPSISESC[0m
    ESC[1mtouch ESC[22m[ESC[4mOPTIONESC[24m]... ESC[4mFILEESC[24m...
ESC[1mDESCRIPTIONESC[0m
    Update the access and modification times of each FILE to the current
    time.

    A FILE argument that does not exist is created empty, unless ESC[1m-c
ESC[22mor ESC[1m-hESC[0m
    is supplied.

    A FILE argument string of - is handled specially and causes touch to
    change the times of the file associated with standard output.

    Mandatory arguments to long options are mandatory for short options
    too.

    ESC[1m-a ESC[22mchange only the access time
    ESC[1m-m ESC[22mchange only the modification time
    ESC[1m-c ESC[22mcreate new files without updating existing files' times
    ESC[1m-f ESC[22mignore nonexistent files and directories, useful only for
    long options
    ESC[1m-d ESC[22mset times using the format DDMMYYhhmm
    ESC[1m-t ESC[22mset times using the format DDMMYYhhmm
    ESC[1m-T ESC[22mset times using the format DDMMYYhhmm
    ESC[1m-s ESC[22mset times using the format DDMMYYhhmm
    ESC[1m-S ESC[22mset times using the format DDMMYYhhmm
    ESC[1m-l ESC[22mcreate symbolic links
    ESC[1m-L ESC[22mcreate symbolic links
    ESC[1m-h ESC[22mcreate hard links
    ESC[1m-H ESC[22mcreate hard links
    ESC[1m-b ESC[22mcreate backup files
    ESC[1m-B ESC[22mcreate backup files
    ESC[1m-k ESC[22mcreate backup files
    ESC[1m-K ESC[22mcreate backup files
    ESC[1m-u ESC[22mcreate backup files
    ESC[1m-U ESC[22mcreate backup files
    ESC[1m-v ESC[22mcreate backup files
    ESC[1m-V ESC[22mcreate backup files
    ESC[1m-w ESC[22mcreate backup files
    ESC[1m-W ESC[22mcreate backup files
    ESC[1m-x ESC[22mcreate backup files
    ESC[1m-X ESC[22mcreate backup files
    ESC[1m-y ESC[22mcreate backup files
    ESC[1m-Y ESC[22mcreate backup files
    ESC[1m-z ESC[22mcreate backup files
    ESC[1m-Z ESC[22mcreate backup files
    ESC[1m-0 ESC[22mcreate backup files
    ESC[1m-1 ESC[22mcreate backup files
    ESC[1m-2 ESC[22mcreate backup files
    ESC[1m-3 ESC[22mcreate backup files
    ESC[1m-4 ESC[22mcreate backup files
    ESC[1m-5 ESC[22mcreate backup files
    ESC[1m-6 ESC[22mcreate backup files
    ESC[1m-7 ESC[22mcreate backup files
    ESC[1m-8 ESC[22mcreate backup files
    ESC[1m-9 ESC[22mcreate backup files
```

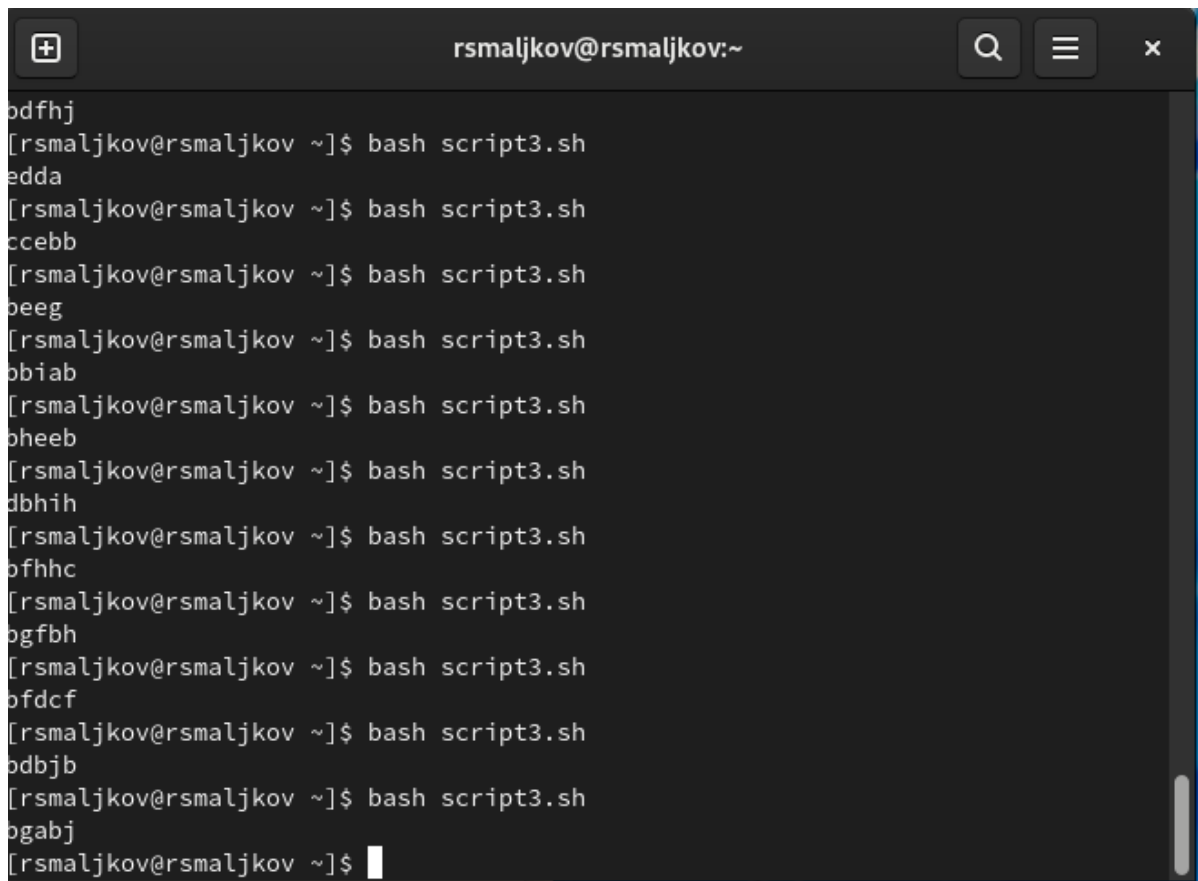
(Скриншот 5)

3. Код и результаты выполнения смотреть в скриншотах 6-7.



```
1 echo $RANDOM | tr '0-9' 'a-zA-Z'
```

(Скриншот 6)



```
odfhj
[rsmaljkov@rsmaljkov ~]$ bash script3.sh
edda
[rsmaljkov@rsmaljkov ~]$ bash script3.sh
ccebba
[rsmaljkov@rsmaljkov ~]$ bash script3.sh
beeg
[rsmaljkov@rsmaljkov ~]$ bash script3.sh
obiab
[rsmaljkov@rsmaljkov ~]$ bash script3.sh
pheeb
[rsmaljkov@rsmaljkov ~]$ bash script3.sh
dbhih
[rsmaljkov@rsmaljkov ~]$ bash script3.sh
ofhhc
[rsmaljkov@rsmaljkov ~]$ bash script3.sh
ogfbh
[rsmaljkov@rsmaljkov ~]$ bash script3.sh
ofdcf
[rsmaljkov@rsmaljkov ~]$ bash script3.sh
odbjb
[rsmaljkov@rsmaljkov ~]$ bash script3.sh
ogabj
[rsmaljkov@rsmaljkov ~]$
```

(Скриншот 7)

Выводы

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