

Отчет по лабораторной работе №12

дисциплина: операционные системы

Шмаков Максим Павлович

Содержание

Цель работы	5
Задание	6
Выполнение лабораторной работы	8
Выводы	15
Контрольные вопросы	16

Список иллюстраций

0.1. рис. 1	8
0.2. рис. 2	9
0.3. рис. 3	9
0.4. рис. 4	9
0.5. рис. 5	10
0.6. рис. 6	10
0.7. рис. 7	11
0.8. рис. 8	11
0.9. рис. 9	11
0.10.рис. 10	12
0.11.рис. 11	12
0.12.рис. 12	13
0.13.рис. 13	13
0.14.рис. 14	14

Список таблиц

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 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 латинско-

го алфавита. Учтите, что \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767.

Выполнение лабораторной работы

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.

Создаю файл `number1.sh` и пишу в нем скрипт. (рис. [-@fig:001]) (рис. [-@fig:002])



```
[mpshmakov@fedora ~]$ touch number1.sh
```

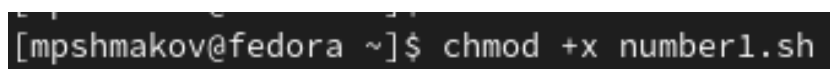
Рис. 0.1.: рис. 1

A screenshot of a code editor window. The title bar shows 'number1.sh' and a file icon. The editor contains a shell script with 14 lines of code. The code uses 'lockfile', 'exec', 'echo', 'until', 'flock', 'do', 'done', 'for', and 'sleep' commands to implement a locking mechanism and a loop.

```
1 lockfile="./lockfile"
2 exec {fn}>$lockfile
3 echo "lock"
4 until flock -n ${fn}
5 do
6     echo "not lock"
7     sleep 1
8     flock -n ${fn}
9 done
10 for ((i=0;i<=5; i++))
11 do
12     echo "work"
13     sleep 1
14 done
```

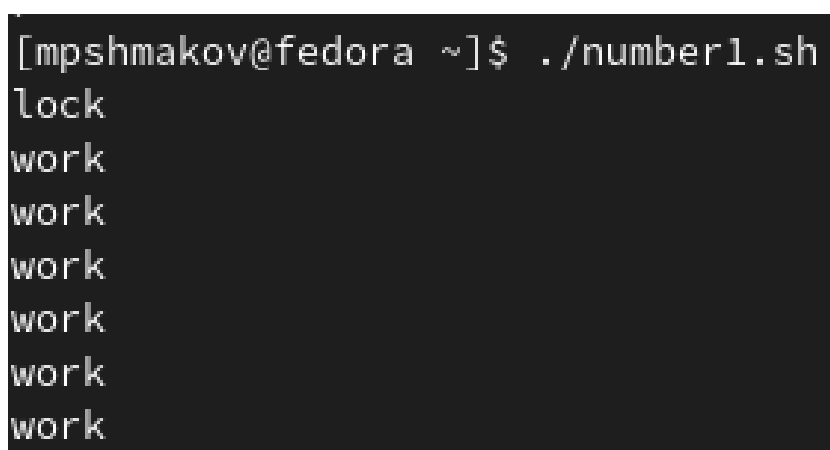
Рис. 0.2.: рис. 2

Даю право на исполнение и проверяю работу скрипта. Все работает правильно. (рис. [-@fig:003]) (рис. [-@fig:004])

A screenshot of a terminal window. The prompt is '[mpshmakov@fedora ~]\$'. The command 'chmod +x number1.sh' has been entered and executed.

```
[mpshmakov@fedora ~]$ chmod +x number1.sh
```

Рис. 0.3.: рис. 3

A screenshot of a terminal window. The prompt is '[mpshmakov@fedora ~]\$'. The command './number1.sh' has been entered and executed. The output shows 'lock' followed by five 'work' lines, each preceded by a newline.

```
[mpshmakov@fedora ~]$ ./number1.sh
lock
work
work
work
work
work
work
```

Рис. 0.4.: рис. 4

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`.

Просмотрел содержимое каталога `/usr/share/man/man1`. (рис. [-@fig:005])

```
[mpshmakov@fedora ~]$ cd /usr/share/man/man1
[mpshmakov@fedora man1]$ ls
:.1.gz
' [.1.gz'
a2ping.1.gz
ab.1.gz
abrt.1.gz
abrt-action-analyze-backtrace.1.gz
abrt-action-analyze-c.1.gz
abrt-action-analyze-ccpp-local.1.gz
```

Рис. 0.5.: рис. 5

Создаю файл `number2.sh` и пишу в нем скрипт. (рис. [-@fig:006]) (рис. [-@fig:007])

```
[mpshmakov@fedora ~]$ touch number2.sh
```

Рис. 0.6.: рис. 6

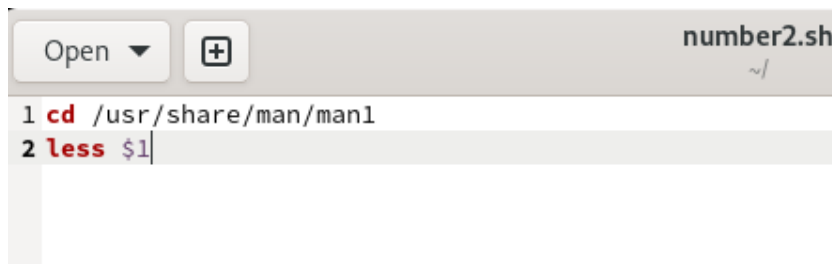


Рис. 0.7.: рис. 7

Даю право на исполнение, запускаю файл и проверяю результат. Все верно.
(рис. [-@fig:008]) (рис. [-@fig:009]) (рис. [-@fig:010])

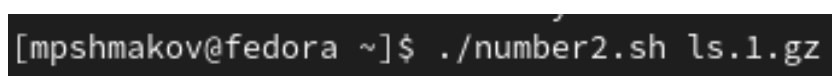


Рис. 0.8.: рис. 8

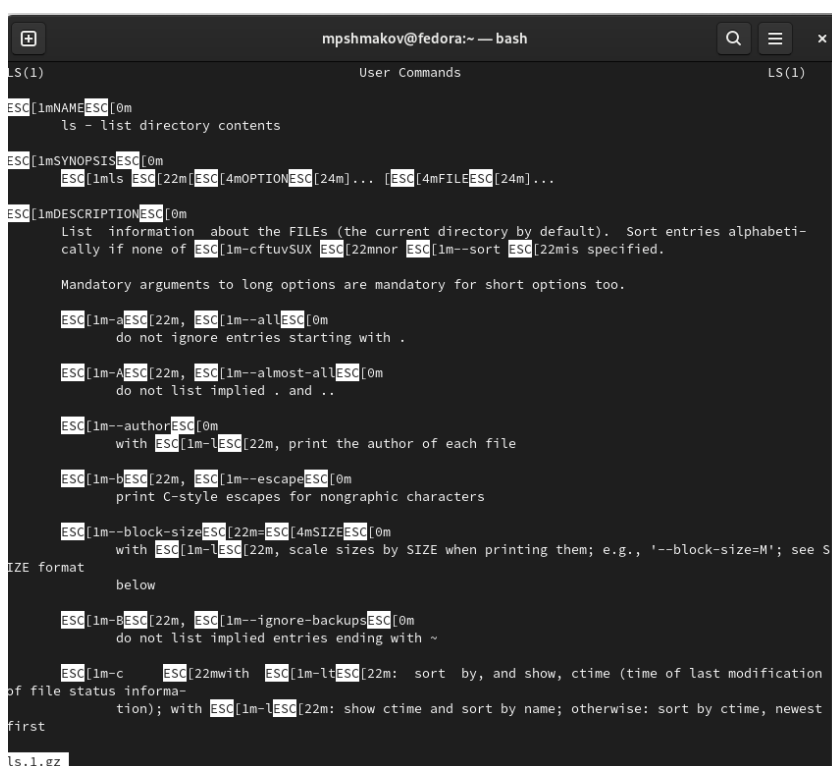


Рис. 0.9.: рис. 9

```
[mpshmakov@fedora ~]$ ./number2.sh ls1.1.gz  
ls1.1.gz: No such file or directory
```

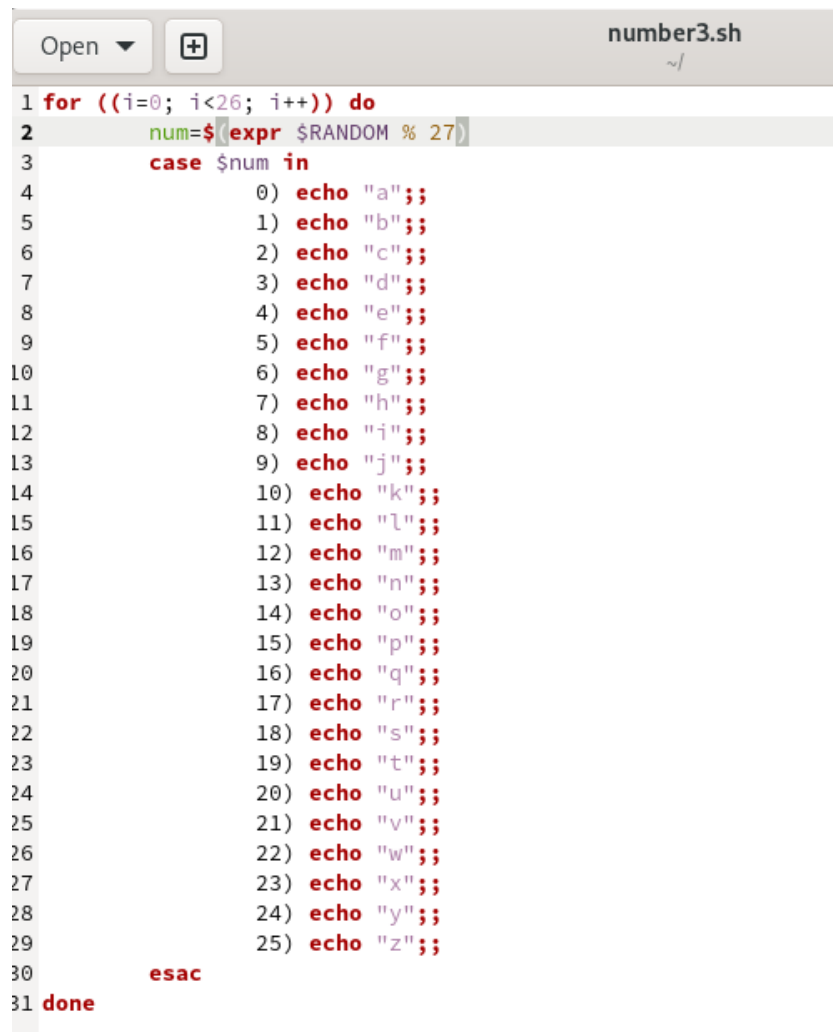
Рис. 0.10.: рис. 10

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.

Создаю файл number3.sh, пишу в нем скрипт. (рис. [-@fig:011]) (рис. [-@fig:012])

```
[mpshmakov@fedora ~]$ touch number3.sh
```

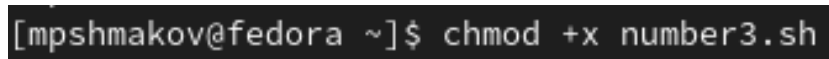
Рис. 0.11.: рис. 11



```
1 for ((i=0; i<26; i++)) do
2     num=$((expr $RANDOM % 27))
3     case $num in
4         0) echo "a";;
5         1) echo "b";;
6         2) echo "c";;
7         3) echo "d";;
8         4) echo "e";;
9         5) echo "f";;
10        6) echo "g";;
11        7) echo "h";;
12        8) echo "i";;
13        9) echo "j";;
14        10) echo "k";;
15        11) echo "l";;
16        12) echo "m";;
17        13) echo "n";;
18        14) echo "o";;
19        15) echo "p";;
20        16) echo "q";;
21        17) echo "r";;
22        18) echo "s";;
23        19) echo "t";;
24        20) echo "u";;
25        21) echo "v";;
26        22) echo "w";;
27        23) echo "x";;
28        24) echo "y";;
29        25) echo "z";;
30    esac
31 done
```

Рис. 0.12.: рис. 12

Даю право на исполнение и проверяю результат. На всякий случай, запускаю 2 раза, чтобы убедиться что выводятся разные буквы. Все верно. (рис. [-@fig:0013]) (рис. [-@fig:0014])



```
[mpshmakov@fedora ~]$ chmod +x number3.sh
```

Рис. 0.13.: рис. 13

```
[mpshmakov@fedora ~]$ ./number3.sh
o
g
p
e
k
n
h
w
s
g
v
a
q
j
b
w
t
q
j
d
j
c
o
g
l
x
[mpshmakov@fedora ~]$ ./number3.sh
e
x
k
q
u
a
y
w
m
l
u
x
o
t
b
```

Рис. 0.14.: рис. 14

Выводы

В ходе работы я научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

Контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке:

```
while [$1 != "exit"]
```

В этой строке квадратные скобки надо заменить на круглые.

2. Как объединить (конкатенация) несколько строк в одну?

Самый простой способ объединить две или более строковые переменные — записать их одну за другой: `VAR1="Hello," VAR2=" World" VAR3="VAR1VAR2"`
`echo "$VAR3"` итог: Hello, World

Вы также можете объединить одну или несколько переменных с литеральными строками:

```
VAR1="Hello," VAR2="VAR1World" echo "VAR2" итог: Hello, World
```

3. Найдите информацию об утилите `seq`. Какими иными способами можно реализовать её функционал при программировании на `bash`?

Команда `seq` выводит последовательность целых или действительных чисел, подходящую для передачи в другие программы.

Мы можем использовать `seq` с циклом `for`, используя подстановку команд, как показано здесь:

```
$ for i in $(seq 1 0.5 4) do echo "The number is $i" done Вывод: The number is 1  
The number is 1.5 The number is 2 The number is 2.5 The number is 3 The number is  
3.5 The number is 4
```


4. Какой результат даст вычисление выражения $\$(10/3)$?

Выдаст результат 3.

5. Укажите кратко основные отличия командной оболочки zsh от bash.

Чтобы дать вам лучше понять набор отличительных черт Z Shell, вот список того, что вы получите, используя Z Shell вместо Bash:

Встроенная команда `zmv` поможет вам массово переименовать файлы/директории, например, чтобы добавить `.txt` к имени каждого файла, запустите `zmv -C '(*)(#q.)' '$1.txt'`. Утилита `zcalc` — это замечательный калькулятор командной строки, это удобный способ считать быстро, не покидая терминал. Загрузите её через `autoload -Uz zcalc` и запустите командой `zcalc`. Команда `zparseopts` — это однострочник, который поможет вам разобрать сложные варианты, которые предоставляются вашему скрипту(?) Команда `autopushd` позволяет вам делать `popd` после того, как вы с помощью `cd`, чтобы вернуться в предыдущую директорию. Поддержка чисел с плавающей точкой (коей Bash, к удивлению, не содержит). Поддержка для структур данных “хэш”. Есть также ряд особенностей, которые присутствуют в Bash, но их нет почти во всех остальных командных оболочках. Вот также некоторые из них:

Опция командной строки `-pois`, которая позволяет пользователю иметь дело с инициализацией командной строки, не читая файл `.bashrc` Использование опции `-rcfile` с `bash` позволяет вам исполнять команды из определённого файла. Отличные возможности вызова (набор опций для командной строки) Может быть вызвана командой `sh` Bash можно запустить в определённом режиме POSIX. Примените `set -o posix`, чтобы включить режим, или `--posix` при запуске. Вы можете управлять видом командной строки в Bash. Настройка переменной `PROMPT_COMMAND` с одним или более специальными символами настроит её за вас. Bash также можно включить в режиме ограниченной оболочки (с `rbash` или `-restricted`), это означает, что некоторые команды/действия больше не будут доступны: Настройка и удаление значений служебных переменных `SHELL`,

PATH, ENV, BASH_ENV Перенаправление вывода с использованием операторов '>', '>|', '<>', '>&', '&>', '»' Разбор значений SHELL_OPTS из окружения оболочки при запуске Использование встроенного оператора eval, чтобы заменить оболочку другой командой И многое другое

6. Проверьте, верен ли синтаксис данной конструкции

```
for ((a=1; a <= LIMIT; a++))
```

Нет, перед LIMIT должен стоять \$.

7. Сравните язык bash с какими-либо языками программирования. Какие преимущества у bash по сравнению с ними? Какие недостатки?

У Bash нет стандартного API, однако он поставляется с простыми встроенными функциями (например, со встроенной тестовой обработкой). Однако вам часто придется создавать процессы для обработки данных. Таким образом, Bash работает очень медленно по сравнению с другими языками, предназначенными для создания автоматизированных сценариев.

Используя встроенные функции Python, можно писать современные сложные Shell-сценарии. Но, в отличие от Bash, интерпретатор Python изначально не поддерживает выполнение процесса.