

## Отчет к лабораторной работе №12

### Common information

discipline: Операционные системы

author: Бабина Юлия Олеговна

group: НПМбд-01-21

### Цель работы

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

### Ход работы

Создадим файл `z1.sh` и откроем его в `emacs`. Напишем 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). Запустим 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. Доработаем программу так, чтобы имелась возможность взаимодействия трёх и более процессов.

```
#!/bin/bash
function ozhidanie
{
    s1=$(date +%s)
    s2=$(date +%s)
    ((t=s2-s1))
    while ((t<t1))
    do
        echo "Ожидание"
        sleep 1
        s2=$(date +%s)
        ((t=s2-s1))
    done
}
function vypolnenie
{
    s1=$(date +%s)
    s2=$(date +%s)
    ((t=s2-s1))
    while ((t<t2))
    do
        echo "Выполнение"
        sleep 1
        s2=$(date +%s)
        ((t=s2-s1))
    done
}
t1=$1
t2=$2
command=$3
```

*код первой программы ч.1*

```

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

```

*код первой программы ч.2*

Проверим корректность работы файла, дав ему права на выполнение, при помощи команды

```
chmod +x z1.sh
```

и запустим его.

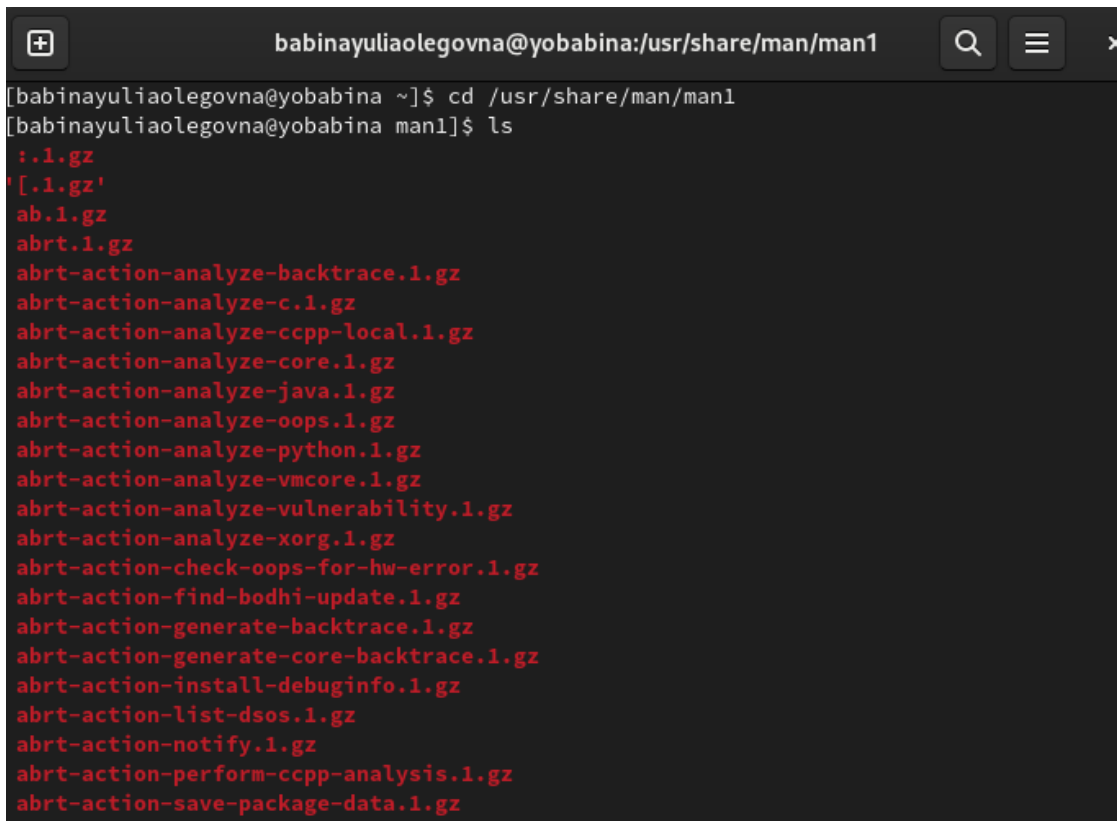
```

[babinayuliaolegovna@yobabina ~]$ ./z1.sh 3 5 Ожидание > /dev/pts/1
bash: /dev/pts/1: Отказано в доступе
[babinayuliaolegovna@yobabina ~]$ ./z1.sh 3 5 Ожидание
Ожидание
Ожидание
Ожидание
Следующее действие:
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Следующее действие:
Выход
Следующее действие:
Выход
Выход
[babinayuliaolegovna@yobabina ~]$

```

*результат выполнения первой программы*

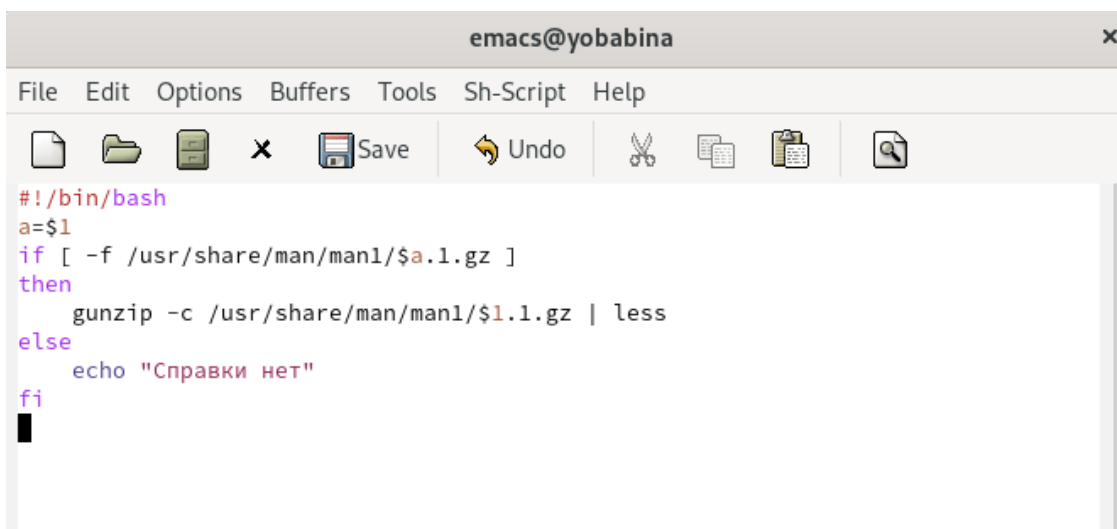
Далее приступим к выполнению задания 2. Сначала изучим содержимое каталога /usr/share/man/man1. В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд.

A terminal window titled 'babinayuliaolegovna@yobabina:/usr/share/man/man1'. The prompt is '[babinayuliaolegovna@yobabina ~]\$'. The user enters 'cd /usr/share/man/man1' and then 'ls'. The output is a long list of .gz files in red text: .1.gz, [.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, abrt-action-analyze-core.1.gz, abrt-action-analyze-java.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-find-bodhi-update.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, and abrt-action-save-package-data.1.gz.

```
babinayuliaolegovna@yobabina:/usr/share/man/man1
[babinayuliaolegovna@yobabina ~]$ cd /usr/share/man/man1
[babinayuliaolegovna@yobabina man1]$ ls
.:.1.gz
' [.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
abrt-action-analyze-core.1.gz
abrt-action-analyze-java.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-find-bodhi-update.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-package-data.1.gz
```

*просмотр каталога*

Реализуем команду ман с помощью командного файла. Для этого создадим файл z2.sh и откроем его в emacs. Напишем скрипт для выполнения задания.

An emacs editor window titled 'emacs@yobabina'. The menu bar includes File, Edit, Options, Buffers, Tools, Sh-Script, and Help. The toolbar shows icons for file operations and editing. The script content is as follows:

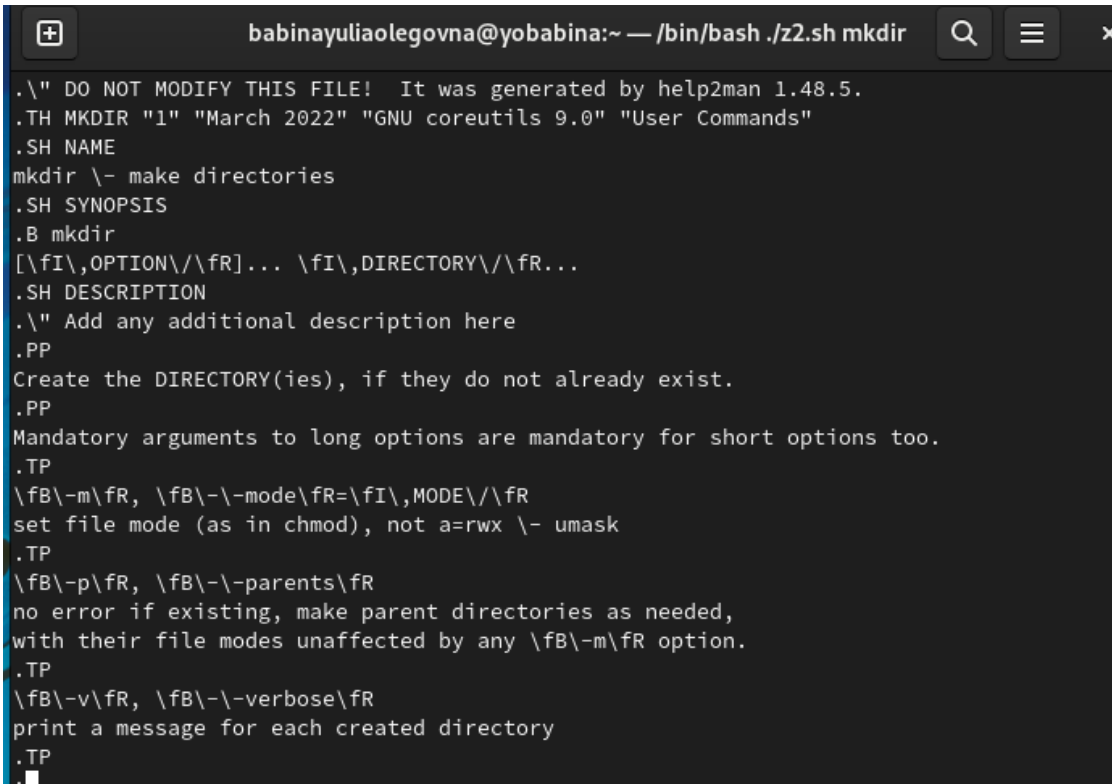
```
#!/bin/bash
a=$1
if [ -f /usr/share/man/man1/$a.1.gz ]
then
gunzip -c /usr/share/man/man1/$1.1.gz | less
else
echo "Справки нет"
fi
```

*код второй программы*

Проверим корректность работы файла (команды `./z2.sh mkdir` и `./z2.sh cat`), предварительно дав ему права на выполнение и запустив его.

```
bash: ./z2.sh: команда не найдена...  
[babinayuliaolegovna@yobabina ~]$ ./z2.sh mkdir  
[babinayuliaolegovna@yobabina ~]$ ./z2.sh cat  
[babinayuliaolegovna@yobabina ~]$
```

*выполнение командного файла*



```
babinayuliaolegovna@yobabina:~ — /bin/bash ./z2.sh mkdir  
.\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.48.5.  
.TH MKDIR "1" "March 2022" "GNU coreutils 9.0" "User Commands"  
.SH NAME  
mkdir \- make directories  
.SH SYNOPSIS  
.B mkdir  
[\fI\,OPTION\/\fR]... \fI\,DIRECTORY\/\fR...  
.SH DESCRIPTION  
.\" Add any additional description here  
.PP  
Create the DIRECTORY(ies), if they do not already exist.  
.PP  
Mandatory arguments to long options are mandatory for short options too.  
.TP  
\fB\-m\fR, \fB\-\-mode\fR=\fI\,MODE\/\fR  
set file mode (as in chmod), not a=rwx \- umask  
.TP  
\fB\-p\fR, \fB\-\-parents\fR  
no error if existing, make parent directories as needed,  
with their file modes unaffected by any \fB\-m\fR option.  
.TP  
\fB\-v\fR, \fB\-\-verbose\fR  
print a message for each created directory  
.TP  
.
```

*результат работы команды mkdir*

```
babinayuliaolegovna@yobabina:~ — /bin/bash ./z2.sh cat
.\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.48.5.
.TH CAT "1" "March 2022" "GNU coreutils 9.0" "User Commands"
.SH NAME
cat \- concatenate files and print on the standard output
.SH SYNOPSIS
.B cat
[\\fI\\,OPTION\\/\\fR]... [\\fI\\,FILE\\/\\fR]...
.SH DESCRIPTION
.\" Add any additional description here
.PP
Concatenate FILE(s) to standard output.
.PP
With no FILE, or when FILE is \-, read standard input.
.TP
\\fB\\-A\\fR, \\fB\\-\\-show\\-all\\fR
equivalent to \\fB\\-vET\\fR
.TP
\\fB\\-b\\fR, \\fB\\-\\-number\\-nonblank\\fR
number nonempty output lines, overrides \\fB\\-n\\fR
.TP
\\fB\\-e\\fR
equivalent to \\fB\\-vE\\fR
.TP
\\fB\\-E\\fR, \\fB\\-\\-show\\-ends\\fR
.
```

*результат работы команды cat*

Далее задание номер 3. Создадим файл z3.sh и откроем его в emacs. Используя встроенную переменную \$RANDOM, напишем командный файл, генерирующий случайную последовательность букв латинского алфавита.

```
emacs@yobabina
File Edit Options Buffers Tools Sh-Script Help
[Icons] Save Undo [Icons] [Icons] [Icons]
#!/bin/bash
a=$1
for ((i=0; i<$a; i++))
do
    ((char=$RANDOM%26+1))
    case $char in
        1) echo -n a;; 2) echo -n b;; 3) echo -n c;; 4) echo -n d;; 5) echo -n e;; 6)
echo -n f;; 7) echo -n g;; 8) echo -n h;; 9) echo -n i;; 10) echo -n j;; 11) echo -n k
;;
        12) echo -n l;; 13) echo -n m;; 14) echo -n n;; 15) echo -n o;; 16) echo -n p;
; 17) echo -n q;; 18) echo -n r;; 19) echo -n s;; 20) echo -n t;; 21) echo -n u;; 22)
echo -n v;;
        23) echo -n w;; 24) echo -n x;; 25) echo -n y;; 26) echo -n z;;
    esac
done
echo
```

*код третьей программы*

Проверим корректность работы файла, дав ему права на выполнение и запустим его.

```
[babinayuliaolegovna@yobabina ~]$ chmod +x z3.sh
[babinayuliaolegovna@yobabina ~]$ ./z3.sh 123
dwxziupwirnufemmqqficiyszifqjndbcjghsehkgsnheajnofeqwqppbwixenexlpxxztnmeaktowcfwfvwdj
yqdejojcginhjykozhcbivptpdqcbapmgqkvf
[babinayuliaolegovna@yobabina ~]$
```

*результат выполнения третьей программы*

## Ответы на контрольные вопросы

### Вопрос 1

В данной строке отсутствуют пробелы после первой скобки и перед последней скобкой. К тому же, не лишним было бы обрамление \$1 в ", так как данная переменная может содержать пробелы.

### Вопрос 2

Самый простой способ объединить две или более строковые переменные — записать их одну за другой:

```
VAR1="Hello, "
VAR2=" World"
VAR3="$VAR1$VAR2"
echo "$VAR3"
```

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

```
VAR1="Hello, "
VAR2="${VAR1}World"
echo "$VAR2"
```

Другой способ объединения строк в bash — добавление переменных или литеральных строк к переменной с помощью оператора +=:

```
VAR=""
for ELEMENT in 'Hydrogen' 'Helium' 'Lithium' 'Beryllium'; do
    VAR+="${ELEMENT} "
done

echo "$VAR"
```

### Вопрос 3

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

Команда seq может пригодиться в различных других командах и циклах для генерации последовательности чисел.

Общий синтаксис команды «seq»:

```
seq [options] specification
```

Например, чтобы просто напечатать последовательность чисел с 1 до 4, можно воспользоваться командой `seq 4`.

В качестве альтернативы ее можно реализовать при помощи цикла `for`:

```
for (( i = $1; i <= $2; i++))do
    echo $i
done
```

#### **Вопрос 4**

Результатом инструкции `$((10/3))` будет 3, так как происходит целочисленное деление.

#### **Вопрос 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`

#### **Вопрос 6**

Синтаксис данной конструкции корректен.

#### **Вопрос 7**

Преимущества `bash`: - Установлен по умолчанию в большинстве дистрибутивах Linux, MacOS - Удобное перенаправление ввода/вывода - Автоматизация некоторых действий с файловыми системами Linux - Работа с серверами

Недостатки `bash`: - Отсутствие дополнительных библиотек - Не является кроссплатформенным языком - Утилиты при выполнении скрипта запускают свои процессы, которые, в свою очередь, отражаются на скорости выполнения этого скрипта.

#### **Вывод**

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