

Отчет по лабораторной работе №12

Операционные системы

Ганина Таисия Сергеевна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	8
4	Выполнение лабораторной работы	9
5	Листинги	14
6	Ответы на контрольные вопросы	16
7	Выводы	19
	Список литературы	20

Список иллюстраций

4.1	Создание и выполнение командного файла	10
4.2	Текст программы	10
4.3	Содержание каталога usr/share/man/man1	11
4.4	Текст программы	12
4.5	Создание и выполнение командного файла	12
4.6	Выполнение программы	12
4.7	Создание и выполнение командного файла	13
4.8	Текст программы	13

Список таблиц

1 Цель работы

Цель данной лабораторной работы - изучить основы программирования в оболочке ОС UNIX, научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

2 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 латинского алфавита.

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

3 Теоретическое введение

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; C-оболочка (или csh) — надстройка на оболочке Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд; оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна; BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C 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/Linux-подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна.

4 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 (рис. 4.1, 4.2)

А
Т
Г
Т
А
С
М
ё

га
И
И
П
Ц
З

Ю
Д

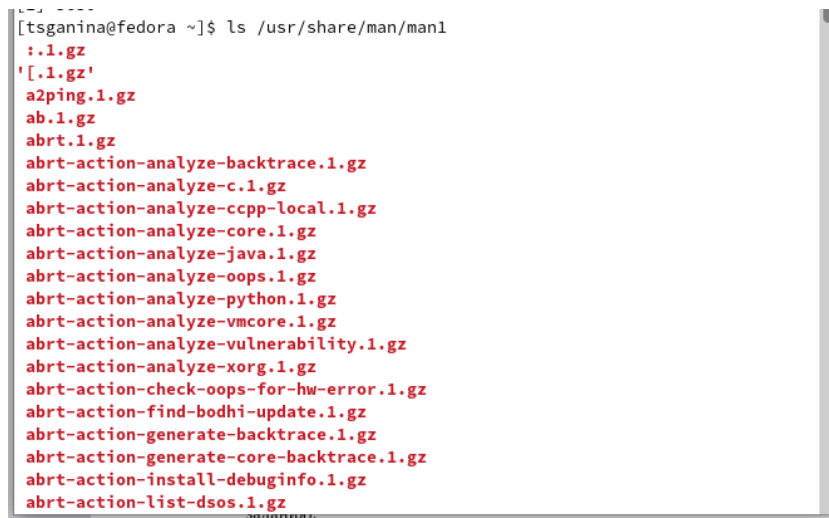
ь
О

Рис. 4.1: Создание и выполнение командного файла

```
1 #!/bin/bash
2 lockfile="./lock.file"
3 exec {fn}>$lockfile
4 while test -f "$lockfile"
5 do
6     if flock -n ${fn}
7     then
8         echo "File is blocked"
9         sleep 5
10        echo "File is unlocked"
11        flock -u ${fn}
12    else
13        echo "File is blocked"
14        sleep 5
15    fi
16 done
```

Рис. 4.2: Текст программы

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` (рис. 4.3, 4.4, 4.5, 4.6)



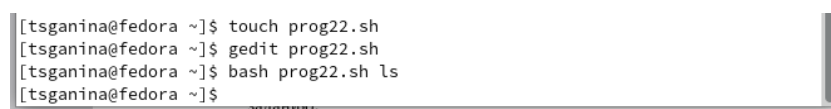
```
[tsganina@fedora ~]$ ls /usr/share/man/man1
.:1.gz
'[:1.gz'
a2ping.1.gz
ab.1.gz
abrt.1.gz
abrt-action-analyze-backtrace.1.gz
abrt-action-analyze-c.1.gz
abrt-action-analyze-ccpp-local.1.gz
abrt-action-analyze-core.1.gz
abrt-action-analyze-java.1.gz
abrt-action-analyze-oops.1.gz
abrt-action-analyze-python.1.gz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

Рис. 4.3: Содержание каталога `usr/share/man/man1`



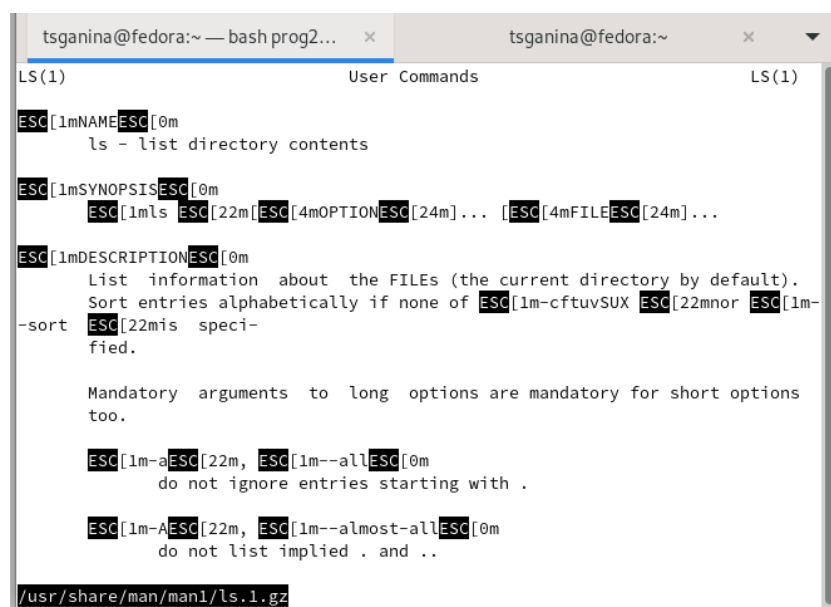
```
1 #! /bin/bash
2 a=$1
3 if test -f "/usr/share/man/man1/$a.1.gz"
4 then less /usr/share/man/man1/$a.1.gz
5 else
6 echo "There is no such command"
7 fi
```

Рис. 4.4: Текст программы



```
[tsganina@fedora ~]$ touch prog22.sh
[tsganina@fedora ~]$ gedit prog22.sh
[tsganina@fedora ~]$ bash prog22.sh ls
[tsganina@fedora ~]$
```

Рис. 4.5: Создание и выполнение командного файла



```
tsganina@fedora:~ — bash prog2... x tsganina@fedora:~ x
LS(1) User Commands LS(1)
ESC[1mNAMEESC[0m
ls - list directory contents
ESC[1mSYNOPSISESC[0m
ESC[1mles ESC[22m[ESC[4mOPTIONESC[24m]... [ESC[4mFILEESC[24m]...
ESC[1mDESCRIPTIONESC[0m
List information about the FILES (the current directory by default).
Sort entries alphabetically if none of ESC[1m-cftuvSUX ESC[22mnor ESC[1m-
-sort ESC[22mis speci-
fied.
Mandatory arguments to long options are mandatory for short options
too.
ESC[1m-aESC[22m, ESC[1m--allESC[0m
do not ignore entries starting with .
ESC[1m-AESC[22m, ESC[1m--almost-allESC[0m
do not list implied . and ..
/usr/share/man/man1/ls.1.gz
```

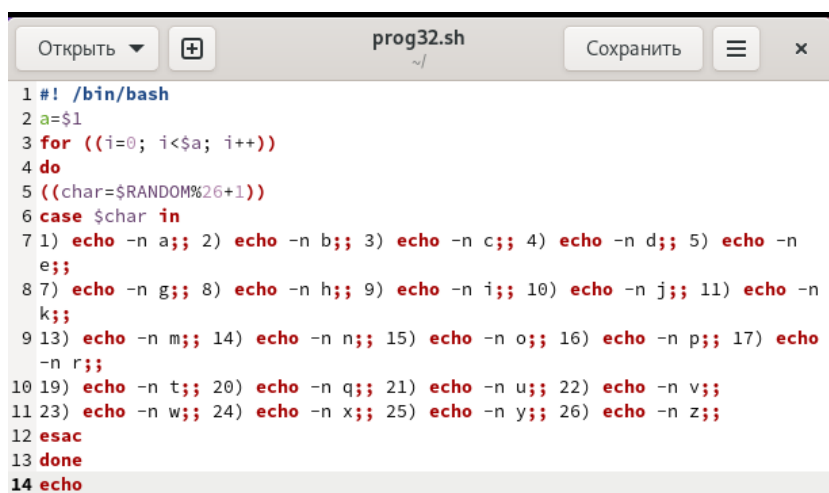
Рис. 4.6: Выполнение программы

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 (рис. 4.7, 4.8)

```
[tsganina@fedora ~]$ gedit prog32.sh
[tsganina@fedora ~]$ bash prog32.sh 2
ab
[tsganina@fedora ~]$ bash prog32.sh 23
jnhumkmnwbxtpgjuum
[tsganina@fedora ~]$ bash prog32.sh 4
bxik
[tsganina@fedora ~]$
```

Рис. 4.7: Создание и выполнение командного файла



```
1 #!/bin/bash
2 a=$1
3 for ((i=0; i<$a; i++))
4 do
5 ((char=$RANDOM%26+1))
6 case $char in
7 1) echo -n a;; 2) echo -n b;; 3) echo -n c;; 4) echo -n d;; 5) echo -n
8 7) echo -n g;; 8) echo -n h;; 9) echo -n i;; 10) echo -n j;; 11) echo -n
9 13) echo -n m;; 14) echo -n n;; 15) echo -n o;; 16) echo -n p;; 17) echo
10 19) echo -n t;; 20) echo -n q;; 21) echo -n u;; 22) echo -n v;;
11 23) echo -n w;; 24) echo -n x;; 25) echo -n y;; 26) echo -n z;;
12 esac
13 done
14 echo
```

Рис. 4.8: Текст программы

5 ЛИСТИНГИ

1. Текст первой программы:

```
#!/bin/bash
lockfile="./lock.file"
exec {fn}>$lockfile
while test -f "$lockfile"
do
if flock -n ${fn}
then
echo "File is blocked"
sleep 5
echo "File is unlocked"
flock -u ${fn}
else
echo "File is blocked"
sleep 5
fi
done
```

2. Текст второй программы:

```
ls /usr/share/man/man1

#!/bin/bash
```

```

a=$1
if test -f "/usr/share/man/man1/$a.1.gz"
then less /usr/share/man/man1/$a.1.gz
else
echo "There is no such command"
fi

```

3. Текст третьей программы:

```

#!/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;;
7) echo -n g;; 8) echo -n h;; 9) echo -n i;; 10) echo -n j;; 11) echo -
n k;
13) echo -n m;; 14) echo -n n;; 15) echo -n o;; 16) echo -n p;; 17) echo -
n r
19) echo -n t;; 20) echo -n q;; 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

```

6 Ответы на контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке: `1 while [$1 != "exit"]`

В данной строчке допущены следующие ошибки: не хватает пробелов после первой скобки [и перед второй скобкой] выражение \$1 необходимо взять в "", потому что эта переменная может содержать пробелы Таким образом, правильный вариант должен выглядеть так: `while ["$1" != "exit"]`

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

Чтобы объединить несколько строк в одну, можно воспользоваться несколькими способами: Первый: `VAR1="Hello," VAR2=" World" VAR3="$VAR1VAR2" echo "$VAR3"`

Результат: Hello, World Второй: `VAR1="Hello," VAR1+=" World" echo "$VAR1"`

Результат: Hello, World

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

Команда seq в Linux используется для генерации чисел от ПЕРВОГО до ПОСЛЕДНЕГО шага INCREMENT. Параметры: seq LAST: если задан только один аргумент, он создает числа от 1 до LAST с шагом шага, равным 1. Если LAST меньше 1, значение is не выдает. seq FIRST LAST: когда заданы два аргумента, он генерирует числа от FIRST до LAST с шагом 1, равным 1. Если LAST меньше FIRST, он не выдает никаких выходных данных. seq FIRST INCREMENT LAST: когда заданы

три аргумента, он генерирует числа от FIRST до LAST на шаге INCREMENT. Если LAST меньше, чем FIRST, он не производит вывод. `seq -f «FORMAT» FIRST INCREMENT LAST`: эта команда используется для генерации последовательности в форматированном виде. FIRST и INCREMENT являются необязательными. `seq -s «STRING» ПЕРВЫЙ ВКЛЮЧЕНО`: Эта команда используется для STRING для разделения чисел. По умолчанию это значение равно /n. FIRST и INCREMENT являются необязательными. `seq -w FIRST INCREMENT LAST`: эта команда используется для выравнивания ширины путем заполнения начальными нулями. FIRST и INCREMENT являются необязательными.

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

Результатом данного выражения $\$((10/3))$ будет 3, потому что это целочисленное деление без остатка.

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

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.

6. Проверьте, верен ли синтаксис данной конструкции `for ((a=1; a <= LIMIT; a++))`

`for ((a=1; a <= LIMIT; a++))` синтаксис данной конструкции верен, потому что, используя двойные круглые скобки, можно не писать \$ перед переменными ().

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

Преимущества и недостатки скриптового языка `bash`: • Один из самых распространенных и ставится по умолчанию в большинстве дистрибутивах Linux, MacOS • Удобное перенаправление ввода/вывода • Большое количество команд для работы с файловыми системами Linux • Можно писать собственные скрипты, упрощающие работу в Linux Недостатки скриптового языка `bash`: • Дополнительные библиотеки других языков позволяют выполнить больше действий • Bash не является языком общего назначения • Утилиты, при выполнении скрипта, запускают свои процессы, которые, в свою очередь, отражаются на скорости выполнения этого скрипта • Скрипты, написанные на `bash`, нельзя запустить на других операционных системах без дополнительных действий.

7 Выводы

При выполнении данной лабораторной работы я изучила основы программирования в оболочке ОС UNIX, научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

Список литературы